Institute for Scientific Research, Inc.

TOWARD RELIABLE NEURAL NETWORK SOFTWARE FOR THE DEVELOPMENT OF METHODOLOGIES FOR INDEPENDENT VERIFICATION AND VALIDATION OF NEURAL NETWORKS

IVVNN-LITREV-F002-UNCLASS-111202

Research Grant NAG5-10269

November 12, 2002

Prepared for:
National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, MD 20771

Attention:
Mr. Harold D. Coleman, Grants Officer
Mr. Nelson H. Keeler, Director, NASA IV&V

Prepared by:
Institute for Scientific Research, Inc.
320 Adams Street
P.O. Box 2720
Fairmont, WV 26555-2720
http://www.isrparc.org

Approval Page

Approved By:	**Ken McGill** NASA Contracting Officer Technical Representative	Date
Approved By:	**Christina Moats** NASA IV&V Project Manager	Date
Approved By:	**Spiro Skias** ISR IVVNN Project Manager	Date
Prepared By:	**Brian J. Taylor** ISR IVVNN Principal Investigator	Date
Prepared By:	**Marjorie Darrah** ISR IVVNN Principal Investigator	Date
Recorded By:	**Karen Tucker** ISR IVVNN Document Controller	Date

EXECUTIVE SUMMARY

Neural networks, are members of a class of software that have the potential to be "intelligent" computational systems capable of simulating characteristics of biological thinking and learning. Owing their origins to the study of the human brain, neural networks possess the ability to acquire and store knowledge. They are well suited for domains of non-linearity and high complexity that is ill-defined, unknown, or just too difficult for standard program practices. Instead of undergoing explicit programming, neural networks adjust themselves to fulfill the need of a desired function.

Intelligent Flight Control is a good example of a system that benefits from neural networks. As technology enables aircraft and spacecraft to perform increasingly complex missions, maintaining control of the crafts becomes comparably more difficult. Consequently, the next generation of flight control systems will utilize adaptive and non-deterministic techniques to provide for more stable and maneuverable aircraft. Neural networks will play a progressively more important role in such systems since they can adapt in real-time to untested flight conditions including aircraft failures, for which engineers are unable to account.

Developers of neural networks have been cautious to extend the use of their applications into safety-critical domains due to the complexities and uncertainties associated with these non-deterministic software techniques. Just as biologists and neuroscientists are hard pressed to understand how the human brain works, mathematicians and computer scientists are also unsure as how an artificial neural network will perform as it undergoes training and adaptation. This raises a concern from project managers and system engineers to the people who will place their trust in these systems: How can we be sure that any system which includes neural network technology is going to behave in a known, consistent and correct manner?

An increased effort on the part of NASA has encouraged research in this area over the past few years, but in general, prior to the 1990s, the study of verification and validation of neural networks has been limited. Universities, government agencies, and a small number of companies are working on differing aspects of this problem, but no single unifying standard or process has been established to help those who are developing neural networks.

The purpose of this report is to summarize methods and tools that may be helpful in developing independent verification and validation for adaptive systems to ensure that they will behave appropriately in safety-critical and high-assurance situations.

Because of the emerging nature of this field, the amount of published material relating solely to the methods and tools that have been used on neural networks in the past is small. In an effort to look for insight, this report also includes summaries of additional methods and tools in related areas that might be adaptable to the independent verification and validation of neural networks (IVVNN).

The information within this document was obtained during an extensive literature survey conducted during a four-month period ending in September 2002. This information base included published research papers (both journal and conference), technical reports, slide presentations, and project and tool documentation. The majority of all material is publicly available through the World Wide Web, a significant part available directly from the authors. In some cases, tool or method developers have been contacted for additional information.

This report first presents a brief background of neural networks and how verification and validation relates to these software systems. It then describes several soft computing systems (though not necessarily neural networks) that illustrate how traditional V&V techniques were used, and how new tools were developed to test nontraditional forms of software. Perhaps the most significant part of this report is the summary of methods. V&V methods that may prove helpful to IVVNN are presented, including rule extraction, model checking, Lyapunov's Direct Method, run-time monitoring, cross validation, improvements in system testing, and visualization. The last section covers potentially useful IVVNN tools. Eighteen tools are described which have been used or developed for adaptive systems review; tool information such as expense, ease of use, automation, available support, and history of performance is presented so that the reader can gain a better understanding of a tool's value. The paper concludes with the anticipated direction of the ISR future efforts on this project.

Although the ISR has identified many tools and methods that aid in the V&V of software systems in general, the team found few that are built for, or applicable to, adaptive systems or neural networks in particular. Of those found, many are still immature techniques that have not been widely used or tested on complex systems. From this literature review, it appears, preliminarily, that the most promising of the methods and tools for V&V of dynamic adaptive neural networks include testing for fixed nonadaptive neural networks and employing an operational monitor. Techniques for formal analysis of artificial neural networks, before, during, and after learning, are probably several years away from proving their effectiveness.

TABLE OF CONTENTS

1.0 INTRODUCTION **1**

2.0 BACKGROUND **3**

2.1 Historical Highlights of Neural Network Development **3**

2.2 History of V&V **6**

2.3 Timeline **8**

2.4 Major Contributors **8**

3.0 LITERATURE SURVEY **10**

3.1 A Summary of Projects **10**

3.1.1 Deep Space Exploration - Remote Agent (RA) 10
3.1.2 Intelligent Aircraft 12
3.1.3 Other Safety-Related Neural Network Projects 15

3.2 Summary of Methods **17**

3.2.1 Testing 19
3.2.2 Run-Time Monitoring 23
3.2.3 Lyapunov Stability Analysis 25
3.2.4 Rule Extraction 26
3.2.5 Cross Validation 37
3.2.6 Visualization 39
3.2.7 Model Checking 46

3.3 Summary of Tools **48**

3.3.1 HyTech 48
3.3.2 Java PathExplorer (JPAX) 49
3.3.3 Java PathFinder (JPF) and Java PathFinder 2 (JPF2) 52
3.3.4 KRONOS 54
3.3.5 LOTOS 55
3.3.6 MATLAB NN Toolbox 57
3.3.7 Murphi – or Murφ 59
3.3.8 PARAGON 60
3.3.9 PAX 61
3.3.10 Planview/Comview 63
3.3.11 PVS 67
3.3.12 Real-Time Software Testing Tool Suite 69
3.3.13 RULEX 71
3.3.14 SMV/NuSMV 74
3.3.15 SPIN 76
3.3.16 STeP 77
3.3.17 UPPAAL 79
3.3.18 WVU F-15 Simulation 82

4.0 EVALUATION OF TOOLS 86

5.0 CONCLUSION 89

5.1 Method Review 89

5.2 Tool Review 90

5.3 Summation 90

6.0 FUTURE RESEARCH 91

6.1 Research on Human Factor Analysis Based on Pilot Certification 91

6.2 Improvements on Existing Processes 91

6.2.1 Formal Methods 91
6.2.2 Run-Time Monitoring Methods 93
6.2.3 Testing Methods 94
6.2.4 Visualization Methods 94

6.3 Tying It All Together 94

7.0 REFERENCES 95

LIST OF TABLES

TABLE 3-1. SUMMARY OF THE FIDELITY LEVELS INTRODUCED 23
TABLE 3-2. RULE EXTRACTION CLASSIFICATIONS 28
TABLE 3-3. SUBSET ALGORITHM 30
TABLE 3-4. M-OF-N ALGORITHM 31
TABLE 3-5. CONJUNCTIVE RULE EXTRACTION ALGORITHM 35
TABLE 3-6. M-OF-N RULE EXTRACTION ALGORITHM 36
TABLE 3-7. FOUR TYPES OF DIVERSITY FOR ANNS 37
TABLE 3-8. SELECTED VISUALIZATION TECHNIQUES 39
TABLE 3-9. MAIN LOTOS OPERATORS 56
TABLE 3-10. RECORD- AND REPLAY- INSTRUMENTATION EXAMPLES 71

LIST OF FIGURES

Figure 2-1. Timeline of Important NN Events 3
Figure 2-2. Diagram of a Biological Neuron 4
Figure 2-3. V&V Literature Timeline 8
Figure 3-1. NASA F-15 ACTIVE Aircraft 13
Figure 3-2. SFDIA Diagram 15
Figure 3-3. Kingsnorth Power Station 16
Figure 3-4. Developmental electronic nose at PNL 17
Figure 3-5. Applicability of Methods for NN Verification and Validation 18
Figure 3-6. Nested Loop Model of Rodvold's Testing Phase 19
Figure 3-7. Diagram of the Trajectory Generator 20
Figure 3-8. Example of Trajectory Clusters 21
Figure 3-9. New Trajectories from Perturbation 22
Figure 3-10. Data Sniffing in an Adaptive System 24
Figure 3-11. Description of pos-att and neg-att 29
Figure 3-12. VIA for a Single Neuron 33
Figure 3-13. Neural Network and Related Hinton Diagram 40
Figure 3-14. Bond Diagram 41
Figure 3-15. Hyperplane Diagram 41
Figure 3-16. Sample Screen from Hyperplane Animator 42
Figure 3-17. Trajectory Diagram 43
Figure 3-18. Hierarchical Structure of Cup Domain Theory 44
Figure 3-19. Lascaux Depiction of a Knowledge-based Neural Network 45
Figure 3-20. Activation signals and effective activation functions 45
Figure 3-21. JPaX Architecture 51
Figure 3-22. Example NN in MATLAB 58
Figure 3-23. Example NN in Simulink 59
Figure 3-24. The PARAGON Toolset 61
Figure 3-25. The Comview Tool 64
Figure 3-26. A Hierarchical View 64
Figure 3-27. The Planview Tool 66
Figure 3-28. PVS Screen Shot 68
Figure 3-29. Language Construct Classified and Scope Analyzed Abstract Syntax Tree 70
Figure 3-30. A 2D Ridge Representative of the Response Area of an LRU 72
Figure 3-31. A 3D Ridge Representative of the Responsive Area of an LRU 72
Figure 3-32. SMV Input File That Models 3-Bit Binary Counter Circuit 75
Figure 3-33. SMV Output Screenshot After Executing Sample Model 75
Figure 3-34. The STeP System Structure 78
Figure 3-35. The STeP User Interface 79
Figure 3-36. UPPAAL2k on Screen. 81
Figure 3-37. The WVU F-15 Simulation 83
Figure 3-38. WVU Menus 84

1.0 INTRODUCTION

This document summarizes the literature survey performed for NASA Goddard Space Flight Center by the Institute for Scientific Research, Inc. (ISR) under the NASA IV&V Center's funded initiative "Development of Methodologies for the Independent Verification and Validation of Neural Networks."

This initiative is the first step toward the goal of developing a new and effective methodology for assessing the artificial intelligence that is increasingly being utilized for space missions as well as commercial, medical, and industrial uses. The overall goal has been separated into six tasks. The first task objectives were to identify, collect, and digest available material related to the verification and validation of neural networks (NNs). This document represents the results achieved from completion of that task.

Independent verification and validation (IV&V) of NN software is of vital importance as the applications for NNs become more feasible and prevalent. Some of the most promising applications of this artificially intelligent technology are in safety-critical situations where NNs can process data and react much faster than a human. One such project is the Intelligent Flight Control (IFC) Program for NASA's F-15 research jet, a project in which the ISR is a partner.

An NN's greatest strength – its adaptability – also creates its greatest challenge: how to assure that its judgment and decisions are sound. This software must be scrutinized to ensure it will perform as expected in every situation. The challenge is that the NN may be adaptive and may encounter unforeseen situations in the field resulting in unpredictable responses. Addressing this challenge will require new methods or adaptations of existing methods.

The goal of the Independent Verification and Validation of Neural Networks (IVVNN) project is to develop a new methodology. This methodology will incorporate the state of the art practices from top researchers in the field. The ISR will apply this new methodology to the NN software used by the IFC project as a proof of concept.

This project focuses on a very specific type of artificial intelligence system: artificial NNs. To clearly understand the goal of developing V&V methods for artificial NNs, it is necessary to define several key terms and concepts.

- An *artificial neural network*, or simply *neural network*, is a computer system that attempts to mimic the way a human brain processes and stores information. It works by creating connections between mathematical processing elements, called *neurons*. Knowledge is encoded into the network through the strength of the connections between different neurons, called *weights*, and by creating groups, or *layers*, of neurons that work in parallel. The system learns through a process of determining the number of neurons or *nodes* and adjusting the weights for the connections based upon training data. In supervised learning, the training data is composed of input-output pairs. An NN tries to find a function which, when given the inputs, produces the outputs. Through repeated application of training data, the network then approximates a function for that input domain.

  This report considers two types of NNs:

  - *Fixed, Non-Adaptive Neural Networks*: Sometimes referred to as a Pre-Trained Neural Network (PTNN), a fixed NN is one that has undergone training and then becomes set. The internal structure of the network remains unchanged during operation. After training is complete, all weights, connections, and node configurations remain the same, and the network reduces to a repeatable function. A common use of a fixed NN might be a classification system to identify malformed products on a manufacturing line where the definition of an undesirable characteristic would not change and the network would be expected to perform the same classification repeatedly.

  - *Dynamic, Adaptive Neural Networks*: Sometimes referred to as an Online Learning Neural Network (OLNN), this type of network is never fixed, so the system continues to develop throughout its life. An OLNN is continuously adapting to current data, changing its internal structure of neurons and weights. OLNNs are employed in situations where a system learns while in use. This is useful where unforeseen scenarios occur, such as aircraft failures, or when input domains change over time, such as stock market analysis.

- *Non-determinism* refers to a property of a computation that may have more than one result. With a deterministic system, logic paths are unchanging: assuming the system is in State S, given Input A, the result is predictably Output B. With non-deterministic systems, the result at different times may be Output B, C, or something else. The result is affected by factors that may not be readily visible to the outside. Neural networks that adapt can be considered non-deterministic because their outputs evolve as learning progresses.
- *Autonomous* means self-controlled or self-directed. Autonomous systems have some onboard intelligence as well as standalone operation and communication capabilities. Neural network structures may, or may not, be part of an autonomous system.

For this summary, the ISR collected and organized more than 300 artifacts concerning NNs and related technologies. These included NASA reports, journal articles, conference papers, presentations, software tools related to the V&V of adaptive systems, abstracts, and assorted literature, which is freely available on the World Wide Web. A process for consideration and digestion of the artifacts was put into place with the following steps:

- Relevance classification
- Artifact indexing
- Summary and review
- Internal presentation and discussion

Because many of these artifacts had only partial relevance to the verification and validation of NNs, they were sorted into relevant and irrelevant categories. Relevant documents were further refined into background and non-background material. Then, relevant artifacts were indexed based upon keywords, category classification, author(s), and other related information, which allowed for some obvious clustering of document groups, such as rule extraction and model checking.

Once indexed, articles were then reviewed and analyzed by members of the ISR project team. Team members created technical summaries, reviews, and critiques of the artifacts. An open discussion on the merits of the artifacts furthered the goal of methodology development.

A list of authors was created as a resource for identifying research activities for future reference. These authors have been contacted about the project and have been invited to contribute their latest research findings. The ISR team has also developed a list of all government agencies, private institutions, and academic institutions that perform research in this area.

From the "fishnet" collection approach, document summaries, and author, organization, and university tracking, the organization of this report emerged.

This report begins with background information, including a brief history of NNs, and of verification and validation and how it relates to these soft computing technologies. Next, a timeline of research is presented, as well as an overview of the major contributors to this and related fields. NASA projects that incorporate autonomous or adaptive software in their design are described, followed by methods and tools that may be used to validate and verify these types of software. A summary table is provided that evaluates the tools and their usefulness toward the goal of performing IVVNN. The document then concludes by offering an approach to unite these methods into an IVVNN practice.

2.0 BACKGROUND

This section offers a brief overview of verification, validation, and NNs. These overviews should not be considered an in depth explanation, but instead a general discussion on each topic to provide background to the methods and tools presented in the next section of this report.

2.1 Historical Highlights of Neural Network Development

This section presents a brief historical summary of major events in the development of NN technology (Figure 2-1) and sets the stage for further consideration of the current state of verification and validation of NNs.

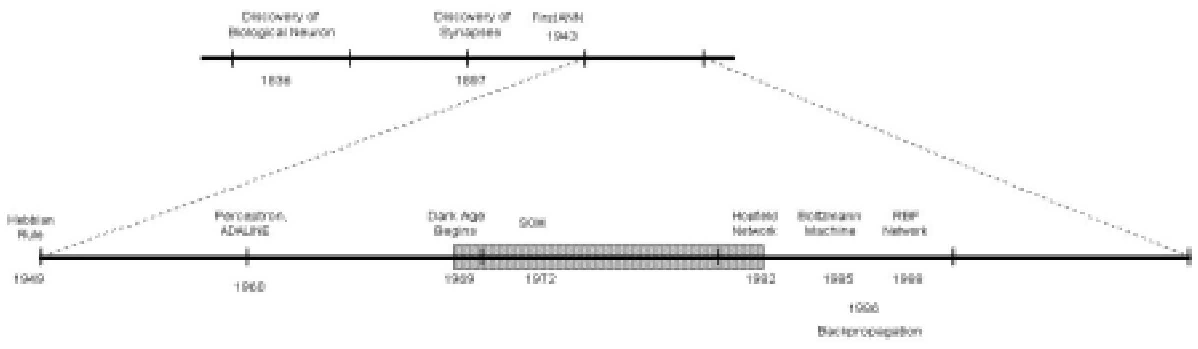


Figure 2-1. Timeline of Important NN Events

Artificial NNs are a mathematical simulation of biological NNs, like the human brain. The basic component of the brain, the neuron, was discovered in 1836. Its structure is depicted in Figure 2-2. In addition to a nucleus, the neuron cell has two specialized appendages: dendrites, which receive impulses from other neurons, and an axon to carry signals to other neurons. The gap between dendrites and axons is called a synapse.

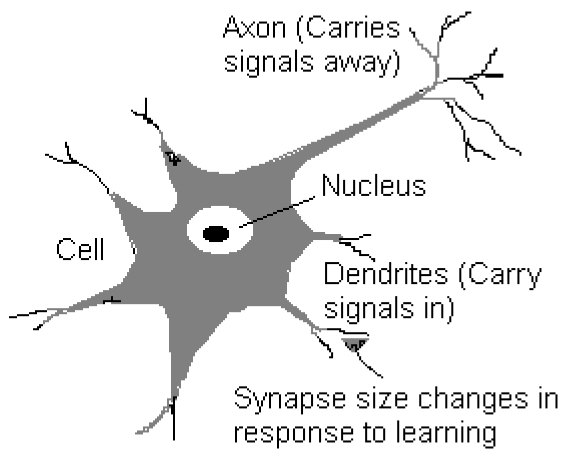

Axon (Carries signals away)
Nucleus
Cell
Dendrites (Carry signals in)
Synapse size changes in response to learning

Figure 2-2. Diagram of a Biological Neuron

Functionally, the neuron acts as a multi-input/single-output unit. A single neuron can have several neighbors connect to it and bring in electrical signals across the synapses and through the dendrites while it alone can connect to one other neuron via the axon. Within the brain, all of the neurons connect to one other via, and work together in, what can be considered a network of neural cells.

The neuron performs a summation of the electrical signals arriving at its dendrites. This summation is compared against a threshold to determine if the neuron shall excite (referred to as firing), resulting in a generation of a signal to the dendrite of another neuron. In 1897, the input signals into a neuron were found to be subject to attenuation in the synapses, meaning the synapses helped to control the strength of the electrical signal passed into the neuron.

The modern era of NN research and development began with the classic work of W.S. McCulloch, a psychiatrist and neuroanatomist, and W. Pitts, a mathematical prodigy, associated with the University of Chicago. With their classic 1943 paper, “A Logical Calculus of the Ideas Immanent in Nervous Activity,” they united the fields of neurophysiology and mathematical logic [McCulloch 1943]. In particular, they showed that a model of a biological NN could, in principle, calculate any computable function.

In 1949, Donald Hebb, a psychologist at McGill University in Canada, published a novel postulate of neural learning: the effectiveness of a synapse to transfer a signal between two neurons is increased by repeated activation across that synapse [Hebb 1949]. This theory, also known as “Hebb’s Rule,” explained the physiological concept of synaptic modification, the increase or decrease of a neuron’s response to electrical stimulus. This corresponds to the use of weighted connections between the neurons of an artificial NN and gave rise to the use of techniques in adjusting these weights during learning.

Hebb’s work influenced Marvin Minsky, who would later go on to found the MIT Artificial Intelligence Laboratory in 1959. While a student at Princeton in 1954, Minsky developed his thesis on “Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem” [Minsky 1954]. Minsky’s book *Computation: Finite and Infinite Machines* [Minsky 1967] extended the 1943 results of McCulloch and Pitts by explaining them in the context of automata theory and the theory of computation.

During this same period, Frank Rosenblatt introduced as a new approach to pattern recognition, the perceptron, culminating in his perceptron convergence theorem [Rosenblatt 1960]. The perceptron represented a significant step over previous attempts at artificial NNs because it introduced the idea of auto-learning frequently occurring patterns. In the same year, Bernard Widrow and Marcian Hoff introduced the least mean-square algorithm and formulated the ADaptive LINear Element (ADALINE) [Widrow 1960]. The ADALINE network used weighting on the inputs into a neuron for pattern classification; it also could take continuous data instead of the predominantly binary inputs used by other networks, including the perceptron.

But even with these new emerging network architectures, the research field was about to collapse. In their book *Perceptrons* [Minsky 1969], Minsky and Seymour Papert mathematically demonstrated some fundamental limitations on single-layer networks like the perceptron. They also expressed their doubt that multi-layer versions could overcome them. These limitations deflated the hype surrounding the great potential of NN technology and led to the decline of continued funding for NN research across the next couple decades (i.e. the "Dark Ages" in Figure 2-1).

Even though interest in NNs waned, there were several researchers still working actively in the field. In the 1970s, von der Malsburg [von der Malsburg 1973] introduced the Self-Organizing Map (SOM). Later, with D.J. Willshaw [Willshaw 1976], he further developed an association of SOMs with topologically ordered maps in the brain. Then in 1980, Grossberg built upon this with a new principle of self-organization known as adaptive resonance theory (ART), which basically involves a bottom-up recognition layer and a top-down generative layer [Grossburg 1980]. Later, in 1982, Tuevo Kohonen introduced the development of SOMs based on one- or two-dimensional lattice structures [Kohonen 1982].

In 1982, J.J. Hopfield introduced the use of an energy function in formulating a new way of understanding the computation performed by recurrent networks with symmetric synaptic connections [Hopfield 1982]. This new perspective, based on energy principles, resulted in attracted many researchers from other scientific disciplines, such as physics, to explore and contribute to the field of NNs. The Hopfield paper also was the first to explicitly make the case for storing information in dynamically stable networks.

In 1983, Kirkpatrick, Gelatt, and Vecchi [Kirkpatrick 1983] introduced a new principle for solving combinatorial optimization problems called simulated annealing, which is rooted in statistical mechanics. Building upon this approach, Ackley, Hinton, and Sejnowski [Ackley 1985] developed a stochastic machine known as the Boltzmann machine, which was the first successful realization of a multilayer NN. This work with the Boltzmann machine provided the foundation for the linking of NNs to belief networks [Pearl 1988] and, in particular, for the development of sigmoid belief networks by Neal [Neal 1992].

In 1986, D.E. Rumelhart and J.L.McClellan, in their monumental two-volume work *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* [Rumelhart 1986], introduced the backpropagation algorithm, which has emerged as the most widely-used learning algorithm for training multilayer perceptrons.

In 1988, D.S. Broomhead and D. Lowe introduced an alternative to multilayer perceptrons with their layered feed forward networks based on radial basis functions (RBF). This work has led to significant efforts to link the design of NNs to the areas of numerical analysis methods and linear adaptive filters [Broomhead 1988].

For a more comprehensive historical analysis of significant achievements in the field of NNs, the reader is referred to the "Historical Notes" section at the end of Chapter 1 in Simon Haykin's *Neural Networks: A Comprehensive Foundation* [Haykin 1999].

2.2 History of V&V

Before software finds its way into safety-critical applications, users of these systems must be assured of highly reliable operation. In non-critical systems, failure may result in loss of work, profits, or mere inconvenience. In systems where high reliability is a requirement, failures can result in massive destruction or loss of human life.

One industry with a high reliability/low failure requirement is aviation. Civilian airliners require highly reliable systems to transport millions of passengers daily. The Federal Aviation Administration, the ruling authority in the U.S., has mandated a failure rate of less than 10^{-9}/hour as the acceptable occurrence of failures within aircraft systems. This means that for every billion hours (roughly 114,000 years) of operation, only one failure should ever occur.

Other industries with high demand for reliability have adopted similar guidelines for acceptable failure rates. Requirements for monitoring systems for nuclear power plants are 10^{-4} failures per hour of operation. The telephone industry commonly cites a limit of 10^{-5} failures per hour. (Customers expect flawless operation from their telephone service provider, so the failure rate is set even higher than the nuclear power industry guidelines.) Phone service should not be interrupted more than two minutes per year, though experience says this is difficult to achieve.

One way to assess the correctness and reliability of a software project is to utilize the practices of verification and validation. V&V methods attempt to answer two questions concerning the entire software lifecycle of a project:

Verification:	Is the product being built right?
Validation:	Is the right product being built?

Verification looks at the end result of the software development process and evaluates the correctness of the software. It seeks to answer questions concerning the adequacy of the processes that went into the system development. Verification also analyzes the outcome of tests conducted on the system that result in metrics that measure the system's expected reliability.

Validation examines the system from a different perspective. Given the original intended uses and needs for the system, and all of the changes and modifications made to those specifications during the software development, does the end product still fulfill those requirements? Validation seeks to ensure that all requirements are met throughout the development of the system. These can include statements on system reliability, failure rates, and other issues important in safety-critical systems.

The software lifecycle can be separated into several stages: concept, requirements, design, implementation, testing, operation, and maintenance. Perhaps due to the visibility of the results from testing, a common misconception is that V&V occurs only during the testing stage. Verification and validation should occur within each stage of the lifecycle. For V&V to be adequate in any kind of system development, each stage must contain its own assurance practices.

The Institute of Electrical and Electronics Engineers published IEEE Standard 1012-1998 (and 1012a-1998) to provide a V&V template for software developers. The *IEEE Standard for Software Verification and Validation* can be used across all processes, activities, and tasks of the software life cycle. The standard identifies key activities that can be conducted within each stage, such as documentation and assessments of risks, hazards, and requirements traceability from stage to stage.

Current V&V techniques, including those described within the IEEE standard, are not well equipped to handle non-deterministic systems like NNs. The use of NNs, especially within safety-critical systems, has been increasing over the past 15 years because they prove very useful in systems that contain ill-defined non-linear functions.

Instead of being programmed and designed in a traditional sense, NNs are “taught” using a learning algorithm and a set of training data. Because of the non-deterministic result of the adaptation, the NN is considered a “black box.” Its response may not be predictable or well defined within all regions of the input space.

Of particular concern is the trustworthiness and acceptability of dynamic NNs that continue to adapt or evolve after the system is deployed. While some OLNNs may be given *a priori* knowledge of their input domain, the adaptation that they undergo offers no guarantee that the system is stable or continues to meet the original objectives.

The V&V technique commonly applied to NNs is brute force testing. This is accomplished by the repeated application of training data, followed by an application of testing data to determine whether the NN is acceptable. Some systems may undergo intensive simulations at the component level, and perhaps at the system level as well. However, these may be no better than “best guesses” toward a system analysis.

In assessing a safety-critical NN system, a V&V expert must know what to look for with an NN and how to analyze the results. Many questions face the analyst regarding the network’s implementation:

- Has the network learned the correct data, or has it learned something else that correlates closely to the data?
- Has the network converged to the global minimum or a local minimum?
- How will the network handle situations when data is presented to it outside of the training set or unique from previous training data?
- Is there a quantifiable metric to describe the network’s “memory” or data retention?
- Is the network making use of the right set of input parameters for the problem domain?

One oft-cited story [Skapura 1996] recounts an NN pattern recognition system that was being developed for the army to identify the presence of enemy tanks. Once trained, the system appeared to work perfectly, able to identify tanks in the testing samples and in a completely separate data set. When taken to the field, however, the system failed. After analysis, it was discovered that the system was actually identifying qualities of the pictures it was being presented with: every photo in the test set that had a tank hidden within it was taken on a cloudy day; coincidentally, every photo without a tank was taken on a clear day. The system had learned to identify cloudy skies and not tanks. This bias had been undetected.

It is stories like this that push the software industry to establish V&V for NN processes. As the development of NNs is often considered more of an art form than a science, so too might it be said about V&V of NNs. Like the IEEE standard, developers need well-defined practices that they can use in their own systems.

2.3 Timeline

The timeline in Figure 2-3 depicts the number of artifacts collected for the different publication years. The artifacts included were relevant towards the development of the methodology for V&V of NNs and consist of articles that pertain to verification and validation of autonomous, adaptive, and non-deterministic systems, as well as NNs. The graph indicates an increase in research publications in this area.

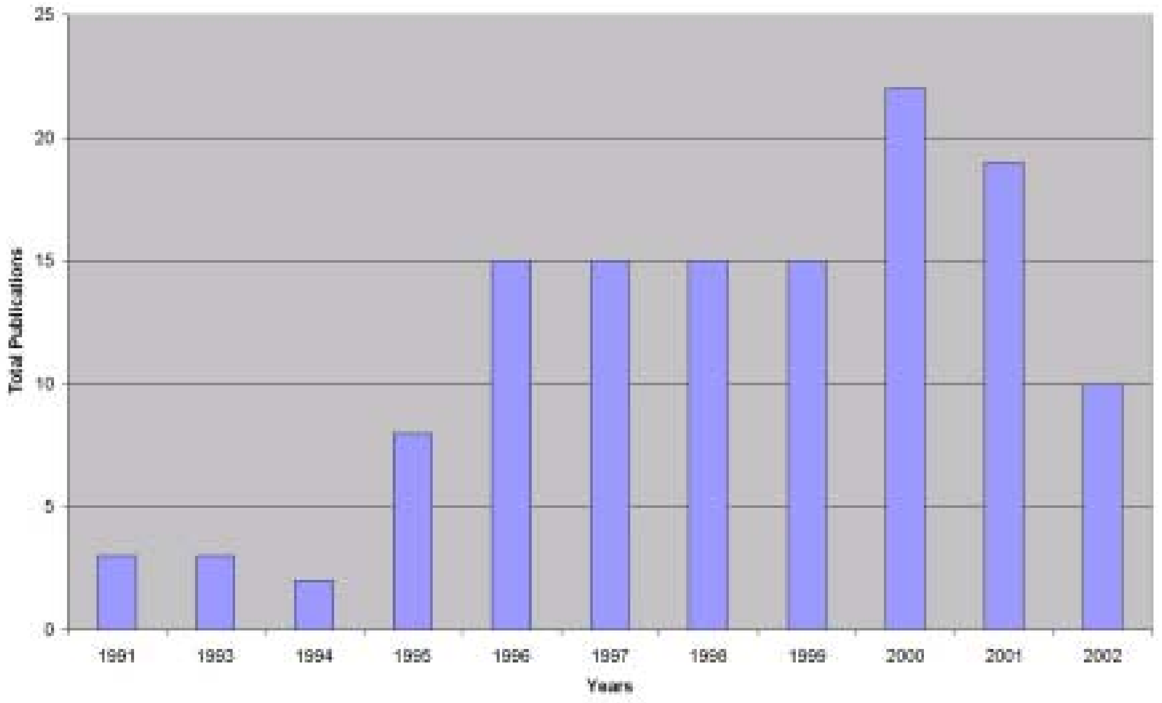


Figure 2-3. V&V Literature Timeline

2.4 Major Contributors

The field of artificial NNs is rapidly evolving. The breadth of problem domains to which they are being applied has expanded considerably in the last few years—including fields as diverse as real-time control systems to general data mining applications, to business predictions, and medical diagnosis. Some programs have potential for a fairly high-level impact in their target industry. In particular, the ISR's own IFC effort could result in significant new opportunities for the application of NN technology within mission-critical arenas such as the aeronautical industry.

Increasingly, the need is being recognized—by such organizations as NASA and the Federal Highway Administration (FHA)—that the supporting systems design function of V&V must be brought to bear for these NN-based systems to gain the necessary acceptance within their respective problem domains.

While many researchers are working in the areas of NN-based systems and general V&V, the number of people, institutions, etc. working in the intersection of these two areas is relatively small. The following paragraphs identify some of the more prominent members of this select group.

NASA is well represented—both directly and indirectly—within this group. Dr. Tim Menzies in the dual role of Software Engineer Research Chair NASA IV&V, and Research Associate Lecturer, Department of Computer Science and Electrical Engineering at West Virginia University, and Dr. Bojan N. Cukic who is an Assistant Professor at West Virginia University, are actively involved in research efforts that impinge upon the V&V of NN-based systems. Charles Pecheur from Research Institute for Advanced Computer Science (RIACS), the Automated Software Engineering Group, and Stacy Nelson, Technology Transfer Consultant, both with NASA Ames Research Center, are also actively involved in R&D relevant to this area of research. Other government agencies, in addition to NASA, have also shown interest in the V&V of NN-based systems.

Larry Medsker, Associate Dean, College of Arts & Sciences & Professor of Physics, The American University, and Rodger Knaus, Principal Investigator at Instant Recall, Inc. (http://www.irecall.com/rkres.htm) in Washington, D.C., have been involved in related efforts for the Federal Highway Administration (FHWA) [Knaus 1998], and are now currently preparing the *FHWA Highway Software Verification and Validation Handbook* on an Instant Recall contract with FHWA.

Several corporate entities are involved in the development of NN-based systems, in general, and for the aeronautical industry, in particular. Accurate Automation Corporation (http://www.accurate-automation.com/) of Chattanooga, TN, and Barron Associates, Inc. (http://www.barron-associates.com/oldsite/index.html) of Charlottesville, VA are two of these. Accurate Automation Corporation (AAC) is a R&D firm specializing in the design and implementation of advanced aircraft technologies. AAC also applies intelligent computing technologies to complex control and signal processing problems in applications such as avionics, robotics, and image processing. AAC's commercial products include the Neural Network Processor (NNP) and Neural Network Tools (NNT). NNP is a high-speed, low-cost processor capable of running complex NNs in real time.

Barron Associates, Inc. (BAI) has incorporated its neural modeling algorithms into an advanced commercial product, GNOSIS, that trains and evaluates artificial NNs for modeling, prediction, estimation, control, and classification purposes. Three particular applications are in the development of 1) an NN-based guidance control for a surface-to-air missile system, 2) on-line adaptive networks for aircraft control, and 3) the verification and validation of fixed-structure NNs for flight critical systems.

In addition to these specialized instances of NN-based systems for aeronautical applications, there are several universities, in addition to WVU, that have significant programs that, while not an exact match, nevertheless, are relevant to the topic of this paper. One such program is led by Reid Simmons [Clarke 2001], Senior Research Computer Scientist at the Robotics

Institute of Carnegie Mellon University, is investigating the application of formal methods to the verification of autonomous systems.

Other groups as far away as Australia also are actively investigating topics relevant to the V&V of NN-based systems. For example, the Advanced Computing Research Centre (ACRC) (http://www.acrc.unisa.edu.au/), School of Computer & Information Science, University of South Australia has two particularly relevant programs:

- ***Neural Computation in a Reconfigurable Computing Environment***, investigating the feasibility of implementing NNs with on-chip learning and making contributions on the best reconfiguration strategies for FPGAs to facilitate real-time adaptation.
- ***Evaluation of Artificial Neural Networks,*** developing new techniques for the cross-validation of artificial NN classifiers.

3.0 LITERATURE SURVEY

The literature survey comprised over three hundred documents, which included publications from conferences, journals, magazines, books, tool and tool documentation and slide presentations. The following sections outline some of the more significant items that were reviewed. The projects section looks at previous programs that made use of NNs and related technology. The methods section highlights seven of the most promising avenues of NN V&V research. The final section investigates currently available tools, some of which deal directly with NNs and others that might be developed further to work with NNs.

3.1 A Summary of Projects

The number of publicly-documented projects that made use of NNs in safety-critical systems appears to be rather small. This emphasizes the fact that these non-deterministic systems lack credibility and confidence and thus far remain untrustworthy for high assurance and high-reliability missions.

Since NNs fall under the classification of non-deterministic systems, there may be potential benefits from examining similar non-deterministic projects such as the Remote Agent (RA) to see if any lessons learned can be extrapolated for NNs. Other projects, such as those found in environmental and medical industries, offer no more discussion on the assessment of NNs beyond repeated network testing. The strongest avenues of NN verification and validation may be those associated with intelligent aircraft. Several of these programs, including the two discussed here, have been ongoing for well over five years and at different points in each program's life cycle, different V&V techniques have been examined.

3.1.1 Deep Space Exploration - Remote Agent (RA)

The RA was the first on-board artificial intelligence system to control an in-flight spacecraft. The control system was created for the launch of Deep Space One, the first flight of NASA's New Millennium program. One of the objectives of the New Millennium program was to increase spacecraft autonomy toward mission-level planning and autonomous health monitoring and recovery. The RA flew the spacecraft between May 17 and May 21, 1999 [Nelson 2002].

Developed by NASA's Ames Research Center (ARC) and Jet Propulsion Laboratory, the RA offered new challenges in design. Because resources on a spacecraft are limited, all activities must be carefully budgeted. The control system must be able to coordinate among multiple activities that may include precise real-time constraints (such as performing an activity at a specific time or event), less-restrictive tasks, and unplanned events. Consequently, such a system must be able to recognize goals and priorities and select, among multiple path options, an optimum path to meet those goals. It must also be able to recognize existing and pending problems and anticipate events. Furthermore, it must be able to do this with limited CPU resources. The resulting product, the RA, was a complex and concurrent software system that employed several automated reasoning engines.

The RA was designed with three layers: a set of core services that comprise a robust operating system, a set of engine modules including a plan runner, and a set of mission-specific task programs. This resulted in three distinct segments: a Planner and Scheduler (PS), a Mode Identification and Recovery (MIR) subsystem, and a Remote Agent eXecutive (RAX).

The PS generated the plans that were implemented by the RA to control the spacecraft. It decomposed goals into task-nets and sequenced the tasks based on precedence and resource constraints. The PS utilized Heuristic Scheduling Testbed System (HSTS) technology, a complex system that elicits and automatically manipulates system level constraints [Nelson 2001]. Items of interest to V&V include tokens, compatibilities, Domain Description Language (DDL), plan model, and plan. Tokens represent intervals of time over which a variable is in a certain state. Compatibilities represent temporal constraints that may involve durations between tokens. DDL is the object-oriented language used for specifying plan models. A plan model is the description of the domain provided in terms of objects and constraints. A plan is "a complete assignment of tokens for all state variables that satisfy all compatibilities, ranges of duration and disjunction of constraints." Formal V&V conducted on HSTS at NASA ARC's Autonomy Group has used the UPPAAL, a modeling, simulation, and verification tool for real-time systems. UPPAAL can represent time and, like HSTS, is a constraint-based system.

The MIR is the model-based health monitoring system (also known as Livingstone) developed at NASA ARC. The Mode Identification module tracks issued commands to estimate the current state of the system; if it varies from the observations from sensors, then it performs diagnosis by searching for the most likely set of component mode assignments consistent with observations. The Mode Recover module then computes a recovery path [Nelson 2001].

The MIR uses a qualitative model of equipment to infer state and diagnose conditions. It observes the RAX, receives state information from the spacecraft, and uses model-based inference to evaluate the state of the craft and to provide feedback to the RAX.

The RAX is the goal-oriented mechanism of the system written in Executive Sequencing Language, an extension of Lisp. It utilizes a concurrent, distributed software architecture that coordinates actions and exchanges information using message passing. It is responsible for the following:

- Requesting and executing plans from the PS
- Requesting and executing failure recoveries from the MIR
- Executing goals and commands from human operators

- Managing system resources
- Configuring system devices
- System-level fault protection
- Achieving and maintaining safe modes as necessary [Nelson 2001]

The RAX has been a testing platform for several V&V efforts. During development, a model of a subset of core services was created using the SPIN model checker. SPIN identified five errors in the program, four of which were classic concurrency errors due to unexpected interleaving. However, SPIN required manual translation to a specialized language named PROMELA.

After the initial success with SPIN, developers sought to create a model checking technology for a mainstream programming language. Early efforts resulted in the translator Java PathFinder, which automated the translation from Java to PROMELA for use in SPIN. When an in-flight error occurred, the RAX was again evaluated in a quick-response, "clean room" experiment testing Java PathFinder. After manually identifying the code sections most likely containing the error, a group of "back end" analysts created a model of the suspicious sections in Java, then used the tool to translate the model from Java to PROMELA for analysis in SPIN. This effort was still labor-intensive, but it led to further advances in tools that reduced manual effort requirements.

3.1.2 Intelligent Aircraft

There are several organizations investigating the use of NNs in aircraft, though the bulk of this work remains in the realms of research and experimental aircraft. The trends for this technology have been to start within research, begin to apply the concepts to military vehicles, and then finally prove the acceptable use of new technology on aircraft.

Two of the projects discussed below, the Intelligent Flight Control program and the Vehicle Health Management program have been conducted at the NASA DFRC. These programs offer a great opportunity for NN V&V because there are several groups, including the ISR, investigating these processes. Research efforts thus far have produced a V&V guidebook [Mackall 2002] for NNs and conference publications.

3.1.2.1 Intelligent Flight Controls

The Intelligent Flight Control (IFC) project is a collaborative effort among the NASA DFRC, the NASA ARC, Boeing Phantom Works, the ISR, and West Virginia University.

Its continuing goal is to develop and flight demonstrate a first generation (GEN1) flight control concept that can efficiently identify aircraft stability and control characteristics using NNs, and utilize this information to optimize aircraft performance in both normal and simulated failure conditions. A secondary goal is to develop the processes to verify and validate NNs for use in flight-critical applications. The flight project results will be utilized in an overall strategy aimed at advancing neural net flight control technology to new aerospace systems designs including civil and military aircraft, reusable launch vehicles, uninhabited vehicles, and space vehicles.

The IFC system will be tested in flight on the NASA F-15 Advanced Control Technology for Integrated Vehicles (ACTIVE) aircraft. This aircraft, shown in Figure 3-1, has been highly modified from a standard F-15 configuration to include canard control surfaces, thrust vectoring nozzles, and a digital fly-by-wire flight control system. The use of canard surfaces along with simulated "stuck" stabilator deflections will allow the program to simulate different actuator failures during flight.

Figure 3-1. NASA F-15 ACTIVE Aircraft

Two types of NNs make up the components to the GEN1 intelligent flight control scheme. A PTNN component provides the baseline approximation of the stability and control derivatives of the aircraft. The PTNN is actually composed of 34 separate multilayer perceptrons, with some of the networks' outputs combined to form the derivatives. The networks were trained with two different training techniques: a modification of Active Selection and the Levenberg-Marquardt algorithm.

The second NN integrated into the IFC system is a highly advanced NN, developed by ARC, named Dynamic Cell Structure (DCS). Flight tests of the OLNN will demonstrate a flight control mode for a damaged fighter or transport aircraft that can return the aircraft safely to base.

Since ensuring pilot and aircraft safety along with overall mission success is a criteria for this program, each of the participating organizations contributed toward the development of a V&V guide [Mackall 2002], Verification and Validation of Neural Networks for Aerospace Systems. This guide was written to assist NASA DFRC in the development of research experiments that use NNs. It is a first approach toward extending existing V&V standards to cover fixed and adaptive NNs.

As of this report, the IFC program continues to test and prepare the program for the first flights using the GEN1 architecture. The next generation of the IFC program is already undergoing development and is simply known as GEN2 at this time. The next generation of IFC includes the use of a higher-order NN known as Sigma-Pi. Instead of providing support for a flight control, this adaptive NN will actually be a component of the controller. Both NASA centers are investigating the use of Lyapunov stability analysis upon the Sigma-Pi network as a method for determining its correctness.

This step-by-step inclusion of NN technologies is the roadmap planned by both NASA DFRC and NASA ARC toward the research and study of NNs within these high safety-critical roles. As the NNs are proven successful, and those assurance methods that are developed provide higher and higher levels of confidence, the roles of the networks become more prominent within the flight control scheme.

3.1.2.2 Intelligent Vehicle Health Management

The sensor failure, detection, identification, and accommodation (SFDIA) system is another area that has utilized NNs in a safety critical role. The goal to reduce physically redundant systems with analytical systems without decreasing vehicle safety or stability is the motivation driving SFDIA development.

Military and civilian aircraft that employ fly-by-wire flight control systems use feedback control techniques to assist in vehicle performance and handling qualities. These control techniques are heavily reliant upon accurate data received from aircraft sensors. Corrupted data have the potential to cause the feedback control laws to command the aircraft into an unstable or unrecoverable flight condition.

The classical solution to reduce such a possibility is to equip the vehicle with two or more physical systems that perform the same function (redundancy). If one system fails or is detected to be untrustworthy, it is deactivated and a duplicate system is activated. The main disadvantages of the classical solution are added weight, additional power requirements, and increased complexity of managing physically redundant systems.

Researchers have begun to study methods to replace the extra physical systems with analytical systems that maintain or increase the level of flight safety associated with them. Toward that goal, Brian Stolarik [Stolarik 2001] utilized extended back propagation NNs in an SFDIA scheme applied to the pitch, roll, and yaw rate gyros of the nonlinear De Havilland 2 aircraft model. Four separate networks were organized in a hierarchal manner. The main NN estimated the roll, pitch, and yaw rate gyro values based on observed aircraft states, while three separate decentralized networks estimated a single parameter (roll, pitch, or yaw) based on different observable aircraft states. An SFDIA flow diagram is provided in Figure 3-2 below.

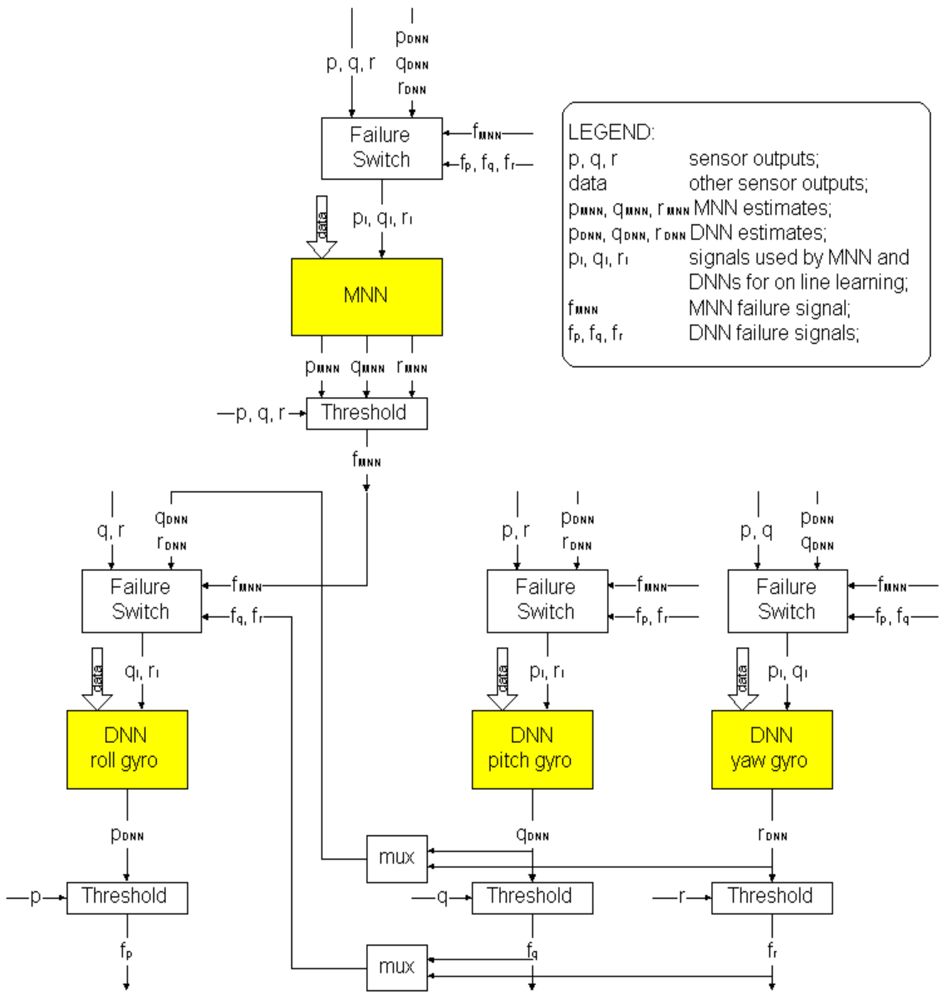


Figure 3-2. SFDIA Diagram

Differences between actual sensor data and those estimated by the various NNs indicated problematic sensors. Once errors occurred the decentralized NN estimate would be substituted for a faulty sensor, thus ensuring quality data for the feedback control system. This method was tested against six types of possible failures to sensor data: large sudden bias, small sudden bias, large fast transient, small fast transient, large slow transient, and small slow transient.

In simulated tests that injected each failure onto each sensor, SFDIA performed as designed by maintaining stable flight.

3.1.3 Other Safety-Related Neural Network Projects

The most visible areas of NN development are found in space-based exploration, autonomous rovers, and military fighter aircraft. While these projects account for the majority of artificial NN press coverage, there are several other areas, all with differing levels of concern for high assurance and safety-related issues. An excellent source for the overview of the state-of-the-art in NN use across commercial industries is found in [Lisboa 2001]. These technologies include financial risk management, chemical detection, medical diagnosis, nuclear reactor control, and manufacturing and process optimization. Some of the projects that deal with safety-critical and safety-sensitive applications are highlighted here.

3.1.3.1 Medical Industry

Neural networks can be applied in the fields of medical diagnosis and decision support systems. As a supporter of medical decisions, not the source of these decisions, they aid health care professionals in improving overall quality of care without the concerns for high safety-critical requirements. One commercially available NN system, Papnet, was developed for the automated classification and to assist in the detection of abnormalities on Pap smears. Studies conducted on the success of Papnet showed that with the system in place, Papnet testing increased the detection of cervical abnormalities by 30% [Lisboa 2001].

Oxford University has developed a high-dependency care device for medical patients who are too ill for normal hospital care but not ill enough for placement in an intensive care unit. This device monitors five physiological parameters including the electrocardiogram, blood pressure, oxygen saturation, respiration, and temperature. Through pattern recognition, the monitor can track physiological instability and alert hospital staff to changes in patient condition.

3.1.3.2 Power Industry

The Generic NOx Control Intelligent System (GNOCIS) is an on-line advisory or closed-loop supervisory system to control the levels of nitrogen oxide emissions from coal-burning power plants. The algorithm works by continuously identifying the optimum settings across several selected plant control variables including coal feeder speeds and/or airflow. The GNOCIS system is an on-line adaptive system; it learns beyond the original data that is installed with the system. Power Technology, which offers specialist engineering and technical services to power plants, reports reductions of up to 15% of NOx emissions while improving boiler efficiency at plants that have the GNOCIS system installed, such as England's Kingsnorth Power Stations shown in Figure 3-3 [Power Technology 1999]. This was a marked improvement over existing physical hardware solutions such as retrofitting existing plants to low-NOx burners.

Figure 3-3. Kingsnorth Power Station

J. Dukelow highlights the problems the nuclear industry faces in accepting the use of NNs in safety critical systems [Dukelow 1994]. Due to concern of the accuracy, predictability, and development of NNs as an art-form, as opposed to rigorous scientific standard, Dukelow argues in favor of increasing V&V methodologies before NNs can assume greater roles within the nuclear industry.

Until that time, it appears that NNs are relegated to an off-line analysis mechanism to improve the results of existing systems. Duke Power and the Knowledge Based Technology Applications Centre (KBTAC) of the Electric Power Research Institute developed an NN application that aids in the evaluation of reactor core control assemblies for wear features. Lisboa explained that such systems, which take on jobs involved in helping human experts, come at the price of low specificity [Lisboa 2001]. This is perhaps an indication that larger roles are desired for NNs but until they can be assured, they must remain in a support role only.

3.1.3.3 Environmental Monitoring

Electronic/artificial noses offer a form of the automated chemical detection and identification technology that are being developed in the commercial sectors. The electronic nose is composed of two basic operations: a chemical sensor array for the collection of molecules, and an artificial NN for the classification of the chemical through pattern recognition. The Pacific Northwest Laboratory (PNL) is undertaking the creation of electronic noses for various uses including toxic waste management, air quality control, and detection of chemical leaks [Keller 1996]. Some electronic nose applications can be safety sensitive [Lisboa 2001]. As law enforcement makes use of these detectors for drug and explosive detection, failure of NN use for classification could result in loss of human life or property damage. Figure 3-4 shows an electronic nose in development at the PNL.

Figure 3-4. Developmental electronic nose at PNL

3.2 Summary of Methods

This summary looks at seven promising areas of research that may lead to the development of standard practices for neural network assurance. These techniques include rule extraction, run-time monitoring, model checking, Lyapunov stability analysis, visualization, improvements in testing methods, and cross validation of NNs. Model checking, which has been applied to non-deterministic systems, such as autonomous software that is multi-threaded, may have less applicability to NNs than other techniques such as rule extraction or run-time monitoring.

Different methods apply to the different stages of the neural network software life cycle (Figure 3-5). Cross validation, a technique that would be employed during the design phase of a project, is realized in the use of ensembles of networks of the same or differing types to accomplish the same task thus improving the dependability of the system. Rule extraction applies best to fixed NNs that have undergone some level of training. If a set of rules that describes how the network will behave has been obtained, the rules can be used in requirements traceability to verify the network against a system specification. Run-time monitoring may best be applied to adaptive NNs and is used after they have been deployed into operation. Lyapunov stability analysis can be used either during a network's development or while it is in operation. Lyapunov stability used during operation would be in conjunction with a system oracle that could decide the NNs performance and make safety judgments. Improved testing techniques, such as automated testing, would be useful after the NN has undergone initialization and training. Visualization proves useful during network training to provide feedback on the learning and decision making processes and as an analysis tool to understand the results of testing.

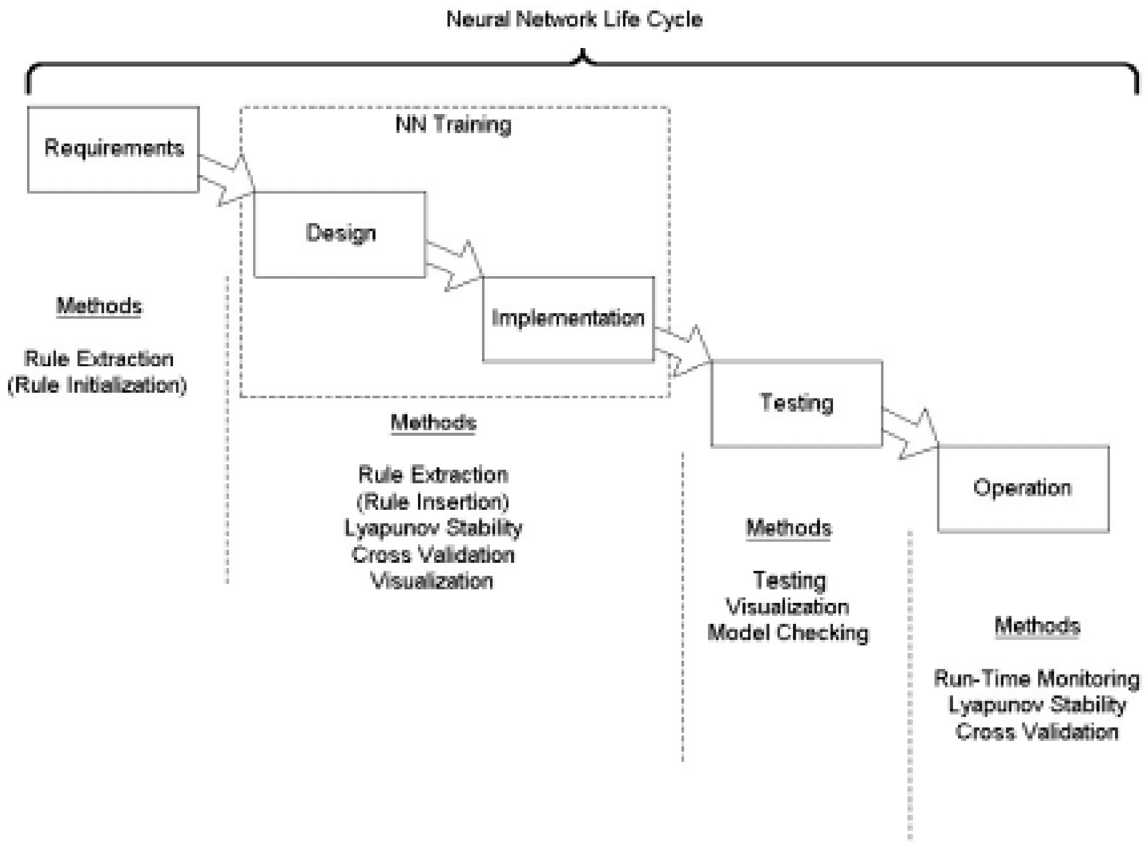


Figure 3-5. Applicability of Methods for NN Verification and Validation

3.2.1 Testing

Testing is the conventional method applied by all NN developers to verify and validate their software. Because NNs are often treated as black boxes, testing becomes a brute-force practice where repeated testing procedures are applied to produce some form of reliability estimation. Neural networks lack the requirement traceability of standard programs and frequently the notion of NN validation focuses more on the correctness of the network within its operational profile than on whether or not it meets parts of a system specification. This section looks at the testing of NNs and how the traditional form of testing NNs may be improved.

3.2.1.1 Traditional NN Testing

The application of traditional software testing methods to IVVNN can be expensive, time consuming, and even unrealistic. Although PTNNs may benefit from traditional testing, in the case of OLNNs these techniques may be inadequate. Traditional testing methods often seek only to prove the functionality and specifications outlined in the requirements phase of development. They do not seek to certify that the software will be stable and function properly after real-time adaptation.

Menzies and Cukic researched the number of tests required to certify traditional software and confirmed that, in certain circumstances, software could be tested sufficiently with only a few randomly selected tests [Menzies 2000]. However, such circumstances were not present often enough to endorse approximate testing for safety critical and mission critical systems.

Rodvold described a software development process model targeted specifically at NNs in critical applications [Rodvold 1999]. This process is illustrated in Figure 3-6. The testing and training phase of the model is comprised of nested loops that perform different functions. *Variation of ANN Topologies* is the innermost loop and it denotes changes of intra-paradigm parameters. Variations in the number of hidden layers or neurons per layer in a multi-layer perceptron are an example of such changes. *Variations of ANN Paradigms* is the next loop and contains more basic network changes such as multi-layer perceptrons or radial-basis functions. *Selection and Combination of ANN Input Neurons* is the outermost loop where changes affect the inputs to be modeled by the NN. For example, this loop would contain any preprocessing of input data needed to reduce noise or minimize dynamic range.

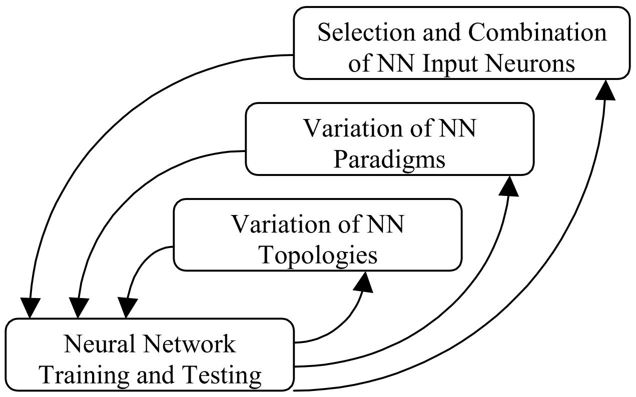

Selection and Combination of NN Input Neurons
Variation of NN Paradigms
Variation of NN Topologies
Neural Network Training and Testing

Figure 3-6. Nested Loop Model of Rodvold's Testing Phase

3.2.1.2 Automated Testing

Automated testing is a practice of reliability assessment that applies a large number of test cases to ensure a good statistical coverage of the input domain. In the aerospace industry where failure rates are expected to be 10^{-9}, the number of failure free tests to achieve a confidence of 99% is on the order of 4.6×10^{9} [Taylor 1999]. Test data generation tools can create large sets of data that can then be used to test such a system. Automated testing employed in cases like this can reduce test development time and the involvement of system testers while returning an increased number of test results.

Most test data generators work on discrete or single valued data. This creates a problem for reliability estimation of real-time systems where inputs consist of linear sequences of data. These sequences of data are defined as data trajectories where each element of the data set is a function of time and a continuous extension over previous data points. Examples of common real-time systems with data trajectories include the measurement of angular accelerations and other sensor data from aircraft, equations describing robotic control, speech analysis, and the data that describes a chemical or nuclear reaction.

A technique developed by Cukic and Taylor is used to generate continuous data for situations where the initial testing set is too small for adequate automated testing [Cukic 2002]. This technique was specifically designed for application to NNs and was used on the SFDIA project described in Section 3.1.2.2.

This trajectory generation scheme relies on expanding existing small sets of test data trajectories to build regressive prediction models that are statistically similar to the original test trajectories. The models establish relationships between independent and dependent variables that allow for perturbation of the independent variables to generate new trajectories from the dependent variables.

The algorithm for this process is separated into two sections as depicted in Figure 3-7, the model generation section and the trajectory generation section. The model generation section consists of collecting a set of test trajectories, processing the data for use by later modules, clustering the test trajectories into a group and developing a regressive model which can best fit the clustered group. Several different regressive models can be used in the developing module, including simple linear, multiple linear, autoregressive, and non-linear regressive models.

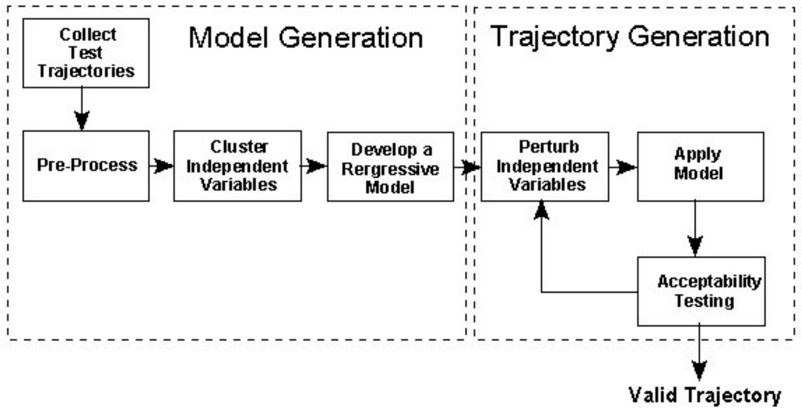

Model Generation
Collect Test Trajectories
Pre-Process
Cluster Independent Variables
Develop a Rergressive Model
Trajectory Generation
Perturb Independent Variables
Apply Model
Acceptability Testing
Valid Trajectory

Figure 3-7. Diagram of the Trajectory Generator

Data can be collected from various sources, including data from actual system usage, data retrieved via a system simulator, or data from previous test cases applied to other similar systems. Because regression predicts the relationship between independent-dependent variables, the collected data must consist of the intended test trajectories to be expanded and additional controllable variables that have correlation to the test trajectories.

By clustering test trajectories into coarse grain regions of the operational profile, regressive models can be defined for each region. For each cluster, a single representative trajectory is selected which best defines the behavior of data within that grouping. From this representative trajectory a regressive model is developed as shown in Figure 3-8.

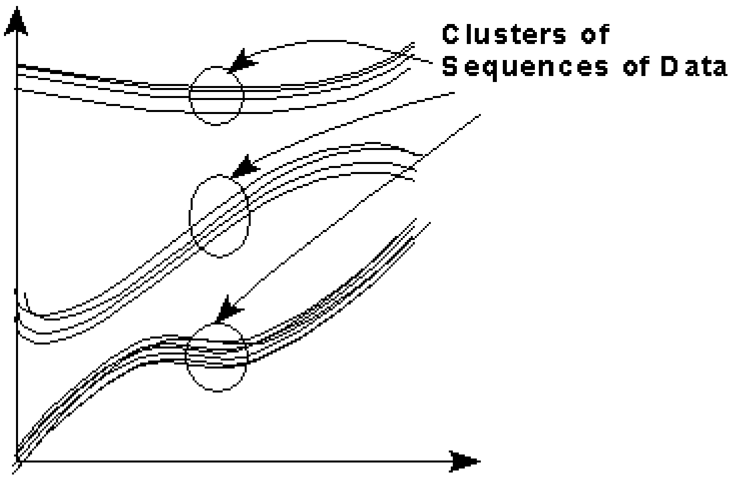


Figure 3-8. Example of Trajectory Clusters

A regressive model is constructed for the representative trajectory and then analyzed to determine fit with the group as a whole. Error calculations consist of applying the regressive model to the remaining trajectories in the cluster looking for a prediction of the dependent variables. Since these dependent variables are already known, an analysis of the correctness for this model within the group can be found. Improvement of the correlation between the predicted and actual trajectories can come from applying a smoothing function to the output of the regressive models or returning to the beginning and selecting a completely different model.

The independent variables from the clusters are then perturbed and the regressive model for that region is applied to the newly created independent variables (Figure 3-9). This method generates a new set of predicted dependent variables, the test data trajectories that are being sought for system testing. Any independent trajectory within the cluster is available to undergo perturbation, even the representative trajectory. It may be beneficial to perturb each of the members of the cluster to create additional sets of test data.

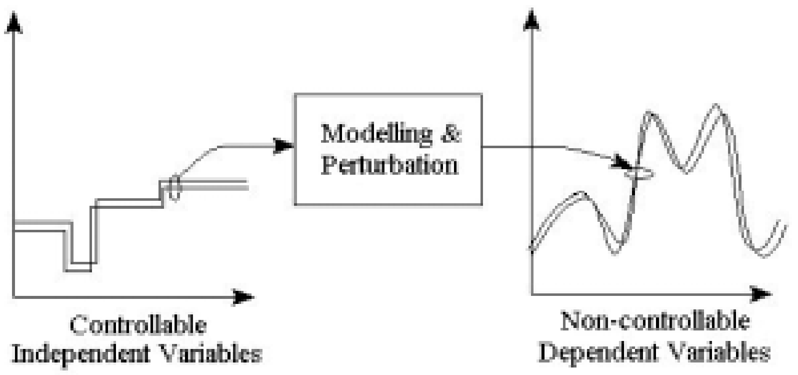


Figure 3-9. New Trajectories from Perturbation

One of the most important parts of the entire process is to determine if the newly created test data actually qualifies as acceptable data. After a set of new dependent variables has been created, it must be examined to ascertain if it can it be used for system testing. A set of rules describing acceptable trajectories can be applied against the predicted trajectories to determine validity. Additional rules can be applied to the perturbed independent trajectories to determine if these new values are acceptable.

This method has been applied to the SFDIA project with promising results that show the trajectory generation algorithm is able to produce new test trajectories faster than they could be simulated or collected from actual usage [Taylor 1999]. A typical simulation flown with a "pilot-in-the-sim" would take about 25 seconds. For the model generation algorithm, a new trajectory was generated every 2.25×10^{-3} seconds, representing a discernible improvement over collection time from standard pilot-in-the-sim data recording. Autoregressive models performed best for prediction of aircraft angular rates with newly generated trajectories being accepted approximately 90% of the time. This implies that the algorithm would generate a large amount of highly acceptable new test cases that could then be applied to automated testing.

3.2.1.3 Simulation

System testing often utilizes a simulation environment wherein the software and hardware interactions are scrutinized. Confidence in the simulation results increases with the fidelity of the simulation platform. A low-fidelity platform, such as a software-only simulator is easy to implement but may not yield the most confidence. A high-fidelity platform, such as one that includes realistic hardware, yields more confidence, but can be expensive, time consuming, and require a high level of expertise. Most testing plans seek to strike an optimal balance between required effort and fidelity in designing simulation environments. This is especially true in those systems that test NNs.

Cukic advocates a modular approach to IV&V of PTNNs, where the fidelity of a simulation platform is assessed in a piecewise fashion and parts are substituted with higher fidelity components until the overall system has reached the highest level of fidelity testing [Cukic 2001]. In the layered approach to testing a PTNN, the focus is placed on the new component after each switch because some level of confidence already exists with the remaining components.

Such an approach is supported by the introduction of fidelity classifications that a simulation platform can have. The classifications range from low fidelity where non-critical components can be tested, to highest fidelity, where safety critical systems are tested on their target platform. The four fidelity levels are outlined in Table 3-1.

System Asset	Fidelity Level	Level Characteristics	Fidelity Sub-Level	Sub-Level Characteristics
Full simulative platform	Low	Control and controlled systems are modeled by software code	Very low	The code can run on the target virtual machine
			Low medium	The code has to be translated before running on the target virtual machine
Hardware in the loop	Medium	A static scaled model of the controlled system is available	Medium low	Inputs provided by a human driver
			Low high	Inputs provided by a computer system
Scalable target system	High	A dynamic scaled model of the controlled system is available		
Target System	Highest	The target system is available		

Table 3-1. Summary of the Fidelity Levels Introduced

3.2.2 Run-Time Monitoring

Online adaptation, such as occurs with non-deterministic NNs, creates unique problems for V&V. Testing at any particular point in time "proves" the system for that moment only; the next data input, whether valid or not, has the potential to alter the behavior of the system as it adapts to accommodate the new information into its behavior. Constant or periodic verification and validation can detect system anomalies before a catastrophic event can occur.

Run-time monitoring involves evaluating the program during execution, and/or evaluating information such as event logs collected from its execution. Generally, this requires a secondary program that runs concurrently with the target application. The run-time monitoring program records specified information (such as data value and timestamp) for selected variables and events. The information collected can be used to detect either violation of system constraints or to manage resources at runtime.

Timing properties can be described as the relationship of events during the execution of a program. Real-time systems can be viewed as a series of events. An event defines the point in time when something happens. Events can be either task events, which denote state changes, or run-time events, which are a set of predefined state changes in the system.

Key components of the run-time monitoring event model framework include:

- An annotation system for specifying events and constraints/assertions to be monitored
- A run-time system for recording and timestamping relevant events
- A satisfiability checker for detection constraint violations at run-time [Jahanian 1995].

In a review of V&V methods for the Integrated Vehicle Health Management project for the Re-usable Launch Vehicle, [Nelson 2002] a lightweight formal method called "the database approach" was acknowledged. In this process, V&V objectives are identified, a database tool is selected, and schema to hold information for analysis is created. The data is parsed (if needed) and loaded. Then database queries are designed based on the V&V objectives.

This method codes artifacts, such as logs, rather than monitoring program execution. It leverages the database as the reasoning engine, providing more coverage than simulation testing, and requires little effort beyond that required of traditional testing.

While traditional run-time monitoring can evaluate the validity of the input and output, "data sniffing" helps assess and control the adaptive reasoning process of the program. This method, illustrated in Figure 3-10, utilizes a pre-alert agent and post-block agent to assess the target program [Lui 2002]. The pre-alert agent captures the incoming data before it enters the system and determines whether or not it may cause unexpected (undesired) adaptations in the system. If so, it offers a warning and allows the data into the system with caution. The post-block agent examines the post-classification value, determines its "distance" to training class norms and, should the new value fall well outside the training domain, the agent prevents it from being used. Distance in early trials was calculated using a *k-means* clustering algorithm (although better algorithms are sought).

In most instances, introduction of such outlier data would not adapt or degrade system performance to unsafe conditions. However, data sniffing would enable an extra layer of protection in the extreme cases where it might.

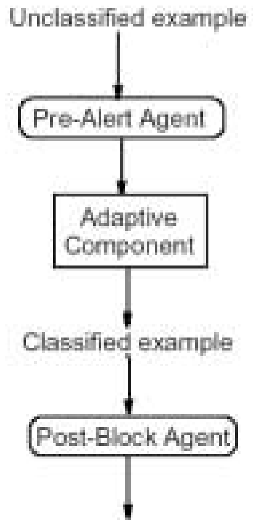


Figure 3-10. Data Sniffing in an Adaptive System

In trials, the data sniffing method worked fairly well for data that is not highly relational. Additional work will include refining the distance calculation algorithm and incorporating existing machine learning tools that detect anomalies in certain data domains.

Benefits of run-time monitoring are that, generally, it requires little incremental effort over traditional testing. It can locate difficult-to-find errors that testers might not find or envision.

Cons include the overhead that it adds to program execution. It is prone to find false positives (problems that do not exist). Furthermore, since run-time monitoring observes generally one execution, certain paths may not be covered in a specific run and, therefore, some errors may be missed.

3.2.3 Lyapunov Stability Analysis

Lyapunov stability analysis can play a critical role in the verification and validation of NNs. Lyapunov's direct (second) method is widely used for stability analysis of linear and nonlinear systems, both time-invariant and time-varying. It can provide insight into a system's behavior without solving the system's mathematical model. Viewed as a generalized energy method, it is used to determine if a system is stable, unstable, or marginally stable.

Nonlinear, time-varying systems, such as NNs, can be mathematically expensive, if even possible, to prove. Used solely as a theorem-proving mechanism, the direct method can guarantee bounded network stability or bounded output. Conclusions about the stability (or instability) of a network can be obtained by constructing a suitable auxiliary function known as a Lyapunov function and denoted as $V(x)$.

Consider an autonomous system described by the differential equation:

$$^{dx}\!/_{dt} = f(x)$$

with an equilibrium point $f(0) = 0$. If a Lyapunov function, $V(x)$, can be determined such that:

$$^{dV(x)}\!/_{dt} \le 0$$

then the equilibrium point is said to be marginally stable. The equilibrium point is considered stable if:

$$^{dV(x)}\!/_{dt} < 0 \tag{3.1}$$

A function $V(x)$ that is either positive definite stable or marginally stable as described above is called a Lyapunov function. It is not unique; rather, many different Lyapunov functions may be found for a given system. Likewise, the inability to find a satisfactory Lyapunov function does not mean that the system is unstable.

There is no universal method for constructing a Lyapunov function. A form of $V(x)$ can be assumed, either as a pure guess or constructed from physical insight. Trial-and-error is normally a last resort.

A better understanding is obtained through an example. Consider the following autonomous system with the equilibrium point (0,0) as detailed in [Boyce 1997]:

$$dx/dt = -x - xy^2$$

$$dy/dt = -y - x^2 y$$

Using the direct method, the stability of the equilibrium point can be discovered. The first step is to describe a Lyapunov function, $V(x)$. The quadratic form is a common starting place:

$$V(x, y) = ax^2 + bxy + cy^2$$

With the proper selection of a, b, and c, $V(x, y)$ can be made to be a positive definite function (greater than zero). Now we need to calculate the rate of change of $V(x, y)$ and substitute the system into it:

$$dV(x,y)/dt = -\left[2a\left(x^2 + x^2y^2\right) + b\left(2xy + xy^3 + x^3y\right) + 2c\left(y^2 + x^2y^2\right)\right]$$

Though this equation is unintuitive, let a and c be any positive number and set $b = 0$. The Lyapunov equation and its derivative now reduce to:

$$V(x, y) = ax^2 + cy^2$$

$$dV(x,y)/dt = -\left[2a\left(x^2 + x^2y^2\right) + 2c\left(y^2 + x^2y^2\right)\right] < 0$$

Since a and c are positive numbers, then regardless of the states of x and y, $V(x, y)$ will be positive and $dV(x, y)/dt$ will be negative. Thus, according to the second condition of Lyapunov's direct method (Equation 3.1), the equilibrium point [Boyce 1997] is stable for this system. A more pertinent example of the use of Lyapunov's direct method on NNs is outlined in [Yu 2002].

3.2.4 Rule Extraction

Rule extraction is the process of developing English-like syntax that describes the behavior of an NN. All rule extraction techniques reviewed share a common prepositional "if...then" format.

Rule extraction offers the possibility of requirements traceability into a system that is not explicitly designed. The rules can also undergo design team review and analysis to detect improper network behaviors or missing knowledge.

Through rule extraction, a system analyst might be able to ascertain novel learning behaviors that had not been previously recognized. By translating these features into a comprehensible English sentence, the analyst can gain not only a better understanding of the network's construction, but perhaps the input domain as well.

Rule Initialization and Rule Insertion

The same techniques used to map rules from the network in rule extraction can also be used in two additional ways: rule initialization or rule insertion.

Rule initialization is the process of giving the adaptive network some pre-system knowledge, possibly through early training or configuration. A system developer may have improved confidence if the starting condition of the network is known, which may lead to a constrained path of adaptation.

Rule insertion is the method of moving symbolic rules back into a network, forcing the network's knowledge to incorporate some rule modifications or additional rules. An adaptive network could benefit from this scheme if the system developer wanted to exert a condition onto the network or reinforce conditions in the network. Examples of this might include restricting the network to a region of the input space or instructing it to deliberately forget some data it has already seen.

For purposes of brevity and conciseness, this review is only concerned with studying how rules can assist in the verification and validation of NNs. However, it should be noted that the other uses for rules could be important towards V&V, especially with regard to controlling how an NN behaves and changes.

Advantages and Disadvantages

Rule extraction from NNs may have greater utility for fixed NNs than for dynamic NNs. Fixed NNs proceed through the steps of training and testing until they reach an acceptable error threshold and only then are they used within a system. The knowledge of the domain is considered embedded inside the weights and connections of the network. If the network is no longer encouraged to adapt, the symbolic rules extracted to describe it can be a useful tool to validate that network at that specific time.

With a dynamic NN, it may be that symbolic rule extraction would be required at intermediate stages in the learning. At some intermediate points symbolic rules would need to be extracted and passed through an oracle or system monitor to confirm that the network was still "correct." It may be that the benefits for dynamic systems lie with rule insertion or rule initialization.

Rule Formats and Definitions

Rule extraction algorithms will generate rules of either conjunctive form or subset selection form, commonly referred to as M-of-N rules named for the primary rule extraction that makes use of the form. All rules follow the English syntactical if-then prepositional form. Conjunctive rules follow the format:

IF condition 1 **AND** condition 2 **AND** condition 3 **THEN** ***RESULT***

Here the RESULT can be of a binary value (TRUE/FALSE or YES/NO), a classification value (RED/WHITE/BLUE), or a real number value (0.18).

The condition can be either discrete (flower is RED, ORANGE or YELLOW) or continuous ($0.25 \leq \text{diameter} \leq 0.6$). The rule extraction algorithm will search through the structure of the network and/or the contents of a network's training data and narrow down values across each input looking for the antecedents (conditions) that make up the rules.

Subset rules, or M-of-N rules, follow the format:

IF (M of the following N antecedents are TRUE) **THEN** ***RESULT***

Cravin and Shavlik explain that the M-of-N rule format provide more concise rule sets in contrast to the potentially lengthy conjunctive rule format [Craven 1994]. This can be especially true when a network uses several input parameters and a value for several of these parameters composes the rule.

3.2.4.1 Rule Extraction Techniques

R. Andrews identifies three categories for rule extraction procedures: decompositional, pedagogical, and eclectic [Andrews 1995a]. Each approach may generate Boolean or fuzzy-logic rules. A breakdown of the classifications and the techniques is presented in Table 3-2.

Table 3-2. Rule Extraction Classifications

	Boolean	**Fuzzy**
Decompositional	Subset KT Rulenet M-of-N KBANN RULEX	Masuoka
Pedagogical	Sato/Nakano VI-Analysis Ruleneg BRAINNE Dedec	Berenji Horikawa FNES (Hayashi) Fune 1 (Halgamuge) fuzzy – MLP (Mitra) Okada
Eclectic	Rule Extraction as Learning	

There are several dozen different rule extraction techniques; many are no more than a succeeding version of a previous technique. The techniques that appear prominently in the literature will be discussed below. Other techniques, such as fuzzy logic and Boolean rule extraction, discussed in the survey paper [Andrews 1995a] do not seem to be widely used or are not well documented, as judged by the lack of information in the literature.

3.2.4.1.1 Decompositional

Decompositional rule extraction involves the extraction of rules from a network in a neuron-by-neuron series of steps. This process can be tedious and result in large and complex descriptions. The drawbacks to decompositional extractions are time and computational limitations. The advantages of decompositional techniques are that they do seem to offer the prospect of generating a complete set of rules for the NN. These rules are also of a binary form; the outputs of the neurons are mapped into a yes/no condition that Andrews refers to as a rule consequent [Andrews 1995a].

Some of the more prevalent decompositional tools and techniques are those discussed by Fu [Fu 1994], Towell and Shavlik [Towell 1993], and Andrews [Andrews 1995a]. They include subset algorithms, M-of-N, RULEX, and RuleNet.

Subset Algorithms

SUBSET and KT are two well-known subset algorithms within decompositional rule extraction.

Fu developed the KT algorithm that is able to handle NNs with a smooth activation function, such as the backpropagation network with a sigmoid function, where the activation function is bounded in the region of [0, 1] [Fu 1994].

The SUBSET algorithm is an extension of the KT algorithm suggested by Towell and Shavlik [Towell 1993]. The SUBSET routine specifies an NN where the output of each neuron in the network is either close to one or close to zero, as opposed to existing somewhere between the bounds of zero and one. This changes the importance of links between neurons in that the values that propagate on a link are close to the value of that link's weights, or zero.

These algorithms make use of the notion of pos-att and neg-att. A pos-att is the input into a node that travels along a positive weight connection. Likewise, a neg-att is an input into a node that is connected with a negative weight. Consider the nodes and connections in Figure 3-11. The value that is passed between node I_1 and O_1 is considered a pos-att, while the value passed between node I_2 and O_1 is a neg-att.

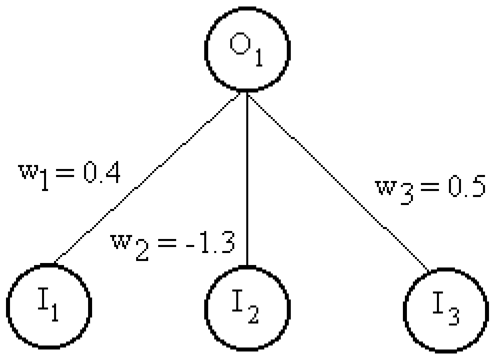


Figure 3-11. Description of pos-att and neg-att

The algorithm proceeds on a node-by-node basis to determine the combination of pos-atts and neg-atts that will activate the node, and from that set creates a rule. The algorithm follows the steps listed in Table 3-3.

Table 3-3. Subset Algorithm

```
For each node in the output layer and each node in a hidden layer
    Search for a set, $S_p$, of pos-atts whose summed weights exceed the threshold of the
    node being investigated
        For each element, $p$, in set $S_p$
            Search for a set, $S_n$, of neg-atts that, in addition to the summed pos-atts from $p$,
            and the summed weights of $n$, from $S_n$, exceed the threshold on the node
                Create a rule, $R_x$, of the form:

            $R_x$: IF $p_1$ AND $p_2$ … AND $p_i$ NOT $n_1$ NOT $n_2$ … NOT $n_j$ THEN RESULT
                where the $p_i$ is the last pos-att in the element $p$ from set $S_p$, $n_i$ is the last neg-
                    att in the element $n$ from set $S_n$, and RESULT is the output from the node.
```

The subset algorithms are capable of finding confirming and disconfirming rules. To find confirming rules, these algorithms look for sets of pos-atts and neg-atts as described above. Confirming rules are composed of pos-att and NOT neg-att combinations which when summed cause the node they feed to activate. Disconfirming rules are found in the same manner. The algorithm looks for combinations of neg-atts and then searches for conjunctions with negative pos-atts.

A drawback to the KT and SUBSET approaches is that the computation time required to find all of the sets of pos-atts and neg-atts is a function of the number of links between nodes and the overall search algorithm is exponential. Fu suggests the use of a control variable, k, to set the size of the rule; k dictates the number of attributes, positive or negative, which may appear in a rule [Fu 1994]. While this has the effect of decreasing computational time, it may cause the algorithm to miss important rules describing the network.

M-of-N

The work of Towell and Shavlik on the SUBSET algorithm led to a rule refinement method on the results of the SUBSET.

The M-of-N method generates rules that are of the form:

IF (M of the following N antecedents are TRUE) **THEN** ***RESULT***

The algorithmic approach for the M-of-N method is reflected in Table 3-4.

Table 3-4. M-of-N Algorithm

1. For each node in the output layer and the hidden layer, form groups of similarly weighted links.
2. Set link weights of all group members to the average of the group.
3. Eliminate any groups that are insignificant to the activation (or non-activation) of the node.
4. Holding all link weights constant, optimize the biases of all hidden and output nodes using the backpropagation algorithm.
5. Form a single rule for each hidden and output node. The rule consists of a threshold given by the bias and weighted antecedents specified by the remaining links.
6. Where possible, simplify rules to eliminate superfluous weights and thresholds.

Towell and Shavlik compare their improvement over SUBSET. They state that because the SUBSET routine generates a larger set of rules than the M-of-N technique, the rule sets returned by M-of-N are usually easier to understand than those of SUBSET. In regards to computation time, the SUBSET algorithm is exponential while the M-of-N technique is approximately cubic. The comparison studies conducted by Towell and Shavlik indicate that the rules generated by M-of-N are approximately equal to the accuracy of the networks that they describe and that the SUBSET algorithm rules are significantly worse.

A potential drawback to the M-of-N approach is that this method requires the network to be knowledge based. (This is a similar drawback to the pre-existing SUBSET and KT methods.)

Andrews [Andrews 1995a] lists the four central requirements for the M-of-N approach based upon the work of Craven and Shavlik [Craven 1994]:

- The NN must be initialized with a rule set or undergo special training to cause a clustering of internal links into equivalence classes,
- The network must have a special training regime,
- Hidden nodes in the network must be approximated as threshold units,
- The extracted rules must use an intermediate term to represent the hidden nodes.

Andrews points out that this may not give rise to a sufficiently accurate description of the network. Further, since the M-of-N approach requires that hidden node meanings not significantly change during the training process, those cases where training does significantly change the internal structure of a network during adaptation may lead to degraded rule sets.

RULEX

RULEX [Andrews 1995b] is a tool and a technique an expanded discussion of RULEX will be given in Section 3.3.13. Created by Andrews and Geva, RULEX is a tool that can extract symbolic rules from a specific kind of multilayer perceptron (MLP), the Constrained Error

Backpropagation (CEBP) network. This network is considered a local function network, but may be more easily understood as a network where the internal structure represents regions of the input space.

Whereas the SUBSET and KT approaches search through the connections of a node to find sets of antecedents, the RULEX approach will generate rules from the weights of the connections between the nodes. This reduces the complexity level of the search for antecedents.

RuleNet

RuleNet is a technique which differs from the above methods in that RuleNet is itself an NN architecture, and by the very nature of the structure and training of the architecture lends itself to rule extraction [McMillan 1991; Andrews 1995a].

RuleNet is constructed of three layers of neurons: an input layer, an output layer, and a middle or hidden layer that the authors call a *condition* layer. The neurons in the condition layer compete in a winner-take-all fashion based upon inputs that pass through the input layer. Each condition-neuron specifies a set of connections from the input layer into the output layer, the output layer then generate the network response. So this intermediary condition layer does not directly contribute to the response, instead it coordinates how the inputs pass into the output layer.

Since each neuron in the condition layer has a chance to be the winner, each neuron operates alone. The extracted rules decompose the weight vector from the condition neuron and the weight vectors from the output layer to form prepositional "if…then…else" rules.

The major drawback to the usage of RuleNet is that it does not lend itself to general techniques; instead it focuses on a specific problem domain.

3.2.4.1.2 Pedagogical

Pedagogical rule extraction is the extraction of a network description by treating the entire network as a black box. In this approach, inputs and outputs are matched to each other. The decompositional approaches can produce intermediary rules that are defined for internal connections of a network, possibly between the input layer and the first hidden layer. Pedagogical approaches usually do not result in these intermediary terms. Pedagogical approaches can be faster than the decompositional, but they are somewhat less likely to accurately capture all of the valid rules describing a network's contents.

Thrun developed Validity Interval Analysis (VI-Analysis or VIA), the core technique within the pedagogical approach [Thrun 1995].

VI-Analysis

Validity Interval Analysis is a pedagogical approach that enables the extraction of rules that directly map inputs to outputs for arbitrary, trained multi-layer backpropagation network [Andrews 1995a].

The key idea of VI-Analysis is to attach intervals to the activation range of each input parameter looking for the network's activations that lie within these intervals. These intervals are called validity intervals. An example of a validity interval on a single input that is bounded by [0, 1] would be [0.2, 0.8]. This is the same as if $0.2 \leq \text{input} \leq 0.8$.

VIA checks whether such a set of intervals is consistent, i.e., whether there exists a set of network activations inside the validity intervals. It does this by iteratively refining the validity intervals, excluding activations that are provably inconsistent with other intervals. The end result is a set of validity intervals for each input, a hypercube across all of the input dimensions.

The rules generated by VIA can take on the form:

IF input $\in$ hypercube$_i$ **THEN** output is ***RESULT***

where a hypercube is the bounded region that encapsulates all of the allowed input values for each input dimension.

There are two phases to determining the validity intervals: forward and backward. The first involves selecting a best guess interval and propagating this interval forward into the network as if it were a single input value.

Consider the single neuron example in Figure 3-12, in which Input I_1 is constrained to the range [0; 0.2] and input I_2 is constrained by the range [0.8; 1.0], and the bias, θ, is -6. The lower box represents what will occur inside the neuron. First the weighted inputs are summed with the bias. This result is acted on by the activation function. Instead of treating the inputs as a single point value, the VIA technique uses the input ranges to compute an output range.

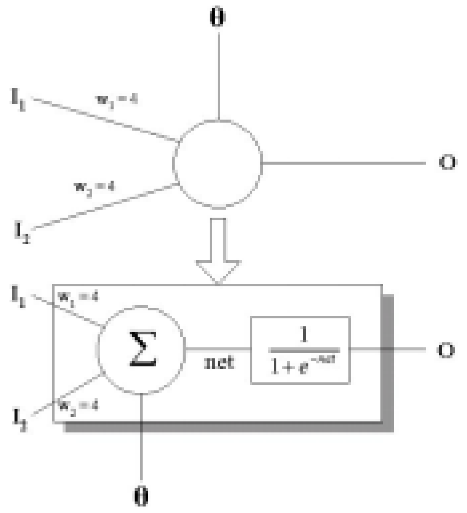


Figure 3-12. VIA for a Single Neuron

The value of net is computed once for the minimum value, then again for the maximum value.

$$net_{\min} = I_1 \cdot w_1 + I_2 \cdot w_2 + \theta = (0 \cdot 4) + (0.8 \cdot 4) - 6 = -2.8$$
$$net_{\max} = I_1 \cdot w_1 + I_2 \cdot w_2 + \theta = (0.2 \cdot 4) + (1 \cdot 4) - 6 = -1.2$$

With the assumption that all transfer functions are continuous, the function can operate upon the net$_{\min}$ and net$_{\max}$ to obtain the output interval.

$$O_{\min} = \sigma(-2.8) = 0.0573$$
$$O_{\min} = \sigma(-1.2) = 0.2315$$

which gives the output interval [0.0573; 0.2315]. This could immediately be mapped into a rule:

$$\textbf{IF } I_1 \le 0.2 \textbf{ AND } I_2 \ge 0.8 \textbf{ THEN } O \in [0.05732;0.2314]$$

assuming the inputs will never exceed the values of 0 or 1, which would occur if they were normalized to the range of [0; 1]. This rule can be simplified further:

$$\textbf{IF } I_1 \le 0.2 \textbf{ AND } I_2 \ge 0.8 \textbf{ THEN } O < 0.5$$

The backward phase of VIA can be conducted when an output interval is known or predicted and all but one of the input intervals are known or predicted. Instead of looking for values that cause an activation, a range for an input is sought based upon working backwards with the activation function and summation.

Staying with the example from Figure 3-12 if an output range for O is known, and an input range for I_1 is known or predicted, then the equations to find the intervals for I_2 would be:

$$I_{2_{\min}} = \frac{\sigma^{-1}(O_{\min}) - (I_{1_{\min}} w_1) - \theta}{w_2}$$

$$I_{2_{\max}} = \frac{\sigma^{-1}(O_{\max}) - (I_{1_{\max}} w_1) - \theta}{w_2}$$

Note that even though this simple example uses one neuron, the strength of the pedagogical approach is that rules are not written in a neuron-by-neuron approach but are built up by traversing the intervals from the outermost layer of the network through to the first layer or vice versa.

The validity intervals are refined using linear programming. An interval is considered fully refined when a set of validity intervals forming the hypercube results in an inconsistency.

Inconsistencies occur on either the input interval with forward propagation or output interval with backwards propagation. The inconsistency is caught when intervals are propagated in a direction and result in a null, or empty, set. For input intervals with forward propagation, an empty set indicates that there is no activation pattern (and thus an NN output) that would satisfy the input interval constraints. For output intervals with backward propagation, the empty set indicates that for the particular activation interval, no input intervals exist that would achieve that output.

There are three reported concerns with this approach. The first is with the linear programming technique employed by VIA in determining if a set of constraints on a network's activations is consistent. Keedwell argues that the linear programming approach is worst-case exponentially complex, even though the results that Thrun publishes never showed this limitation [Keedwell 2000]. A second is that the repeated application of the linear programming algorithm to NNs with large numbers of neurons may increase the solution time to an exceptionally long duration. The third concern is on the generated rules themselves. Duch points out that the VIA technique may have a tendency to extract rules that are too specific and rather numerous [Duch 2001].

3.2.4.1.3 Eclectic

The *eclectic* approach is merely the use of those techniques that incorporate some of a decompositional approach with some of a pedagogical approach, or techniques designed in such a way that they can be either decompositional or pedagogical. The Rule-extraction-as-learning method, for example, is designed such that it can use either technique.

Rule-Extraction-As-Learning (REAL)

In a paper describing the REAL technique, Cravin and Shavlik discuss a way of extracting rules through supervised learning and network querying as opposed to the common search-based techniques from the previous sections [Craven 1994]. (They refer to the search methods as Rule-extraction-as-search approaches.) Many of the search algorithms try to find rules that explain the activations of hidden layer and output layer neurons in the networks. The REAL technique instead will learn from the training examples and query a network to determine if the specific instances from the training set are covered by the target output result.

Craven and Shavlik argue that the search-based methods require a computational complexity that is exponential to the number of input parameters to the network. For some situations, their REAL method can be proven to have lower computational requirements and yet arrive at rule sets that have the same degree of accuracy.

Table 3-5 outlines the RULE algorithm from Craven and Shavlik. The REAL system uses two oracles; an EXAMPLES oracle and a SUBSET oracle. The EXAMPLES oracle produces training examples (from the existing training set) for the rule-learning algorithm. The authors proposed that when the EXAMPLES oracle exhausts the contents of the training set, it might be possible to randomly generate new examples (as might be done by an automated test generation algorithm). The SUBSET oracle takes the arguments of a class label (the output result) and a conjunctive rule and tries to determine if all of the instances covered by the rule are members for that target output. If so, SUBSET returns true and continues. If not, then this input into SUBSET serves as the basis for a new rule.

Table 3-5. Conjunctive Rule Extraction Algorithm

```
/* initialize rules for each class */
for each class c
     $R_c$ := Ø
repeat
     e := EXAMPLES()
     c := classify(e)
     if e not covered by $R_c$ then
          /* learn a new rule */
          r := conjunctive rule formed from e
          for each antecedent $r_i$ of r
               r' := r with $r_i$ removed
               if SUBSET(c, r') = *true* then r := r'
          $R_c$ := $R_c$ V r
until stopping criterion met
```

Once a new rule is found, the algorithm attempts to generalize it by removing antecedents from the rule until a minimum set of antecedents can be combined into a conjunctive statement.

SUBSET can be implemented as either a pedagogical approach such as VI-Analysis, or as a decompositional approach where the rules are extracted for each hidden and output neuron individually.

The authors also made a version of REAL that extracted M-of-N rules since these rules are typically more concise than conjunctive rules.

[Craven 1994] explain that the M-of-N REAL algorithm works in the same manner as the conjunctive rule extraction in Table 3-5. The algorithm, seen expanded in Table 3-6, first learns a conjunctive rule using an example from the training set supplied by the EXAMPLES oracle. The algorithm then converts the conjunction into a M-of-N rule where M = N and the antecedent is merely placed into the set. The next step is to generalize the rule by applying two operators:

- Add-value: adds a new feature into the set of antecedents that is not yet present
- New-term: takes an existing set of antecedents and splits it into two sets of the form L-of-L and (M-L)-of-(N-L).

The SUBSET oracle is responsible for determining if the generalized rule is consistent with the network. The algorithm continues looping through all operator applications until the rule can no longer be generalized.

Table 3-6. M-of-N Rule Extraction Algorithm

```
/* initialize rules for each class */
for each class c
     $R_c$ := Ø
repeat
     e := EXAMPLES()
     c := classify(e)
     if e not covered by $R_c$ then
          learn conjunctive rule, r, as in Table 3-5
          trivially convert r to a M-of-N rule where M = N
          do
               r′ = result of applying add-value or new-term to r
               if SUBSET(c, r′) = true then r := r′
          while ∃ additional operator applications
          $R_c$ := $R_c$ V r
until stopping criterion met
```

The test studies suggest that the REAL conjunctive method is two orders of magnitude faster than the search based conjunctive methods and the REAL with M-of-N performed an order of magnitude slower than REAL with conjunctive rules and an order of magnitude over search-based conjunctive rules methods. The decreased speed using the M-of-N approach did lead to more concise rules.

For networks with a small search space, the search-based methods may be best. For networks with a large number of features, the learning method will be best.

One of the main drawbacks to this algorithm is that it does not yet deal with real-valued output features. This paper focused on discrete-valued output features, which limits its applicability to these domains.

3.2.5 Cross Validation

Cross validation is an approach similar to N-version programming. Its basis is "reliability through redundancy," a concept from software engineering. The cross validation concept centers on combining diverse Artificial Neural Networks (ANNs) into an ensemble. The output of the component networks may then be checked against one another to affirm validity and appropriateness. This model is particularly suitable for safety-critical environments.

While results indicate that using ensembles of NNs increases the performance over a single NN, redundancy alone does not ensure improved performance. It must first be determined what kind of diversity may lead to improved performance, and what is the best way of creating sets of NNs that show this kind of diversity [Sharkey 1995a].

Amanda and Noel Sharkey list numerous methods for such ensemble combinations. One such method relies on training NNs from different starting points, or different initial conditions. Another method varies the topology or number of hidden units, or the algorithm involved. Other methods rely on varying the data, such as sampling, using different data sources, different preprocessing methods, distortion, and adaptive resampling.

If the ensemble is made up of differing types of networks, then the output (prediction) of each of the NNs is collected and combined. The combination can be done in several ways, including voting, average, or weighted average. This method is used to improve the output of an NN by having several "opinions" on which to base the final decision [Krogh 1995].

The Sharkeys discuss four types of diversity for ANNs. Type 1 and Type 2 guarantee reliability with simple majority vote, Type 3 improves reliability with sophisticated selection that may create reliable ANN systems, and Type 4 can improve reliability but cannot lead to a fully reliable ANN system. Features of each type are presented in Table 3-7.

Table 3-7. Four types of diversity for ANNs

Type	Features
1	• The target function is covered by the collection of ANNs • None of the failures are coincident • Majority will always be correct since only one will ever fail on the randomly chose input • Requires n > 2
2	• The target function is covered by the collection of ANNs • Allows coincident failures • Majority is always correct • Requires n>4
3	• The target function is covered by the ANNs • Majority voting is not always correct • More sophisticated voting is performed, weights are assigned to the output of different nets
4	• The target function is not covered by the ANNs • There are failures shared by all ANNs

Two methods have been presented to create the diversity necessary for NN ensembles to be efficient.

- Use different sample sets to train each copy of the ANNs. This increases diversity, but has not been shown to achieve Type 1 and Type 2 diversity.
- Use different interpretations of input states from a system to train the ANNs. This is also known as contrasting measures methodology. For example, two contrasting measures taken from a ships engine state, such as temperature and pressure, may be used as training input to different groups of ANNs. They are effectively being trained on different functions both being used for the same purpose. This is the most statistically independent among the solutions.

A study by Sharkey, Sharkey and Chandroth uses the combination of varying sample sets and contrasting measures to create a Type 1 system [Sharkey 1995b].

Applications

These methods have been applied in several studies, including engine health monitoring, a launch interceptor problem, and a compare-length problem [Sharkey 1995a].

The first study on engine health monitoring investigated the possibility of creating a Type 1 system using three ANNs for online detection and diagnosis of combustion faults in a marine engine. The ANNs were trained to classify pressure data generated by the MERLIN[1] engine simulator. The same sample of 150 training pairs was used to train each of the three ANNs. One ANN was trained on the raw data, and the other two were trained on preprocessed data using two different kinds of transformations. A test set of 414 pairs were reserved. The ANN trained on the raw data exhibited a 99.3% correct generalization on the test set. The ANN trained on the data after preprocessing with Transformation A exhibited 98.1% correct generalization. The one trained on the data preprocessed under Transformation B exhibited a 95.7% correct generalization. Although the three ANNs exhibited high performance, none were 100% accurate. When the three were combined into a voting configuration, they covered the function, they encountered no coincident failures, and the majority vote was always correct (at least on the test set). Therefore, this transformation methodology can lead to Type 1 diversity.

The second case study, a launch interceptor, showed similar results. Three ANNs were trained on sets of 500 data values, one with raw data, and the other two with transformed data. The goal was to classify the data as a *Launch* or *No Launch* example. Five thousand examples were reserved as a test set. In this study, the ANNs each performed above 98%. When the networks were combined to form a system, they had no coincident errors, they covered the function, and the majority vote was always correct for the test set. This is another example of Type 1 diversity.

In the third test case, compare-length function, an ANN was trained on 9,408 patterns. A test set of 8,280 patterns was created. The ANN exhibited a 99.8% correct generalization for the test set. Since this was a geometric problem, rotating the input space through six different angles created diversity. The six angles of rotation and their percentage correct generalizations on the test set were 50° (99.49%); 100° (99.65%); 150° (99.52%); 200°

1 Engine simulation software developed by Lloyds Register of Shipping, London.

(99.43%); 250° (99.65%); 300° (99.75%). The diversity created by rotating the test set was of Type 2 because there were coincident failures. However, in this situation, the majority was still always correct on the test set.

Pros and Cons

The advantage to using the cross validation method of combining ANNs is the ability to increase the performance of an NN system by introducing diversity. The disadvantage is that the component NN still requires verification and validation.

Cross validation must be built in at the design phase of the project.

3.2.6 Visualization

Understanding the operation of NNs is no small undertaking. Neural networks for solving real-world problems may have several thousand connections. Understanding the representations formed by the network during the learning process requires making sense of a vast amount of real-valued parameters. Furthermore, network units usually have many incoming connections.

For designers and end-users of NNs to have confidence in the performance of the system, however, they must understand how it arrives at its decisions. Visualization helps bridge these cognitive chasms by illustrating relationships and flows.

Scientific visualization involves transforming data into visual forms that can be easily understood. Humans have highly developed abilities for visual pattern recognition that can be capitalized when vast quantities of data are transformed into a qualitatively different form. Visualization can provide insight into both the decision-making process and the learning process of NNs.

Visualization may assist NN users in discovering data features whose importance was not previously recognized. It may also help in understanding changes to the system that have occurred during training. These techniques also allow the user to detect error or patterns more easily because they appear as visual anomalies. Additionally, visualization software can provide an interactive mechanism that enables the user to adjust parameters and quickly see the effects of the changes.

Craven and Shavlik discuss several visualization techniques [Craven 1992]. These techniques provide insight into the decision-making processes and the learning processes of NN. These techniques are listed in Table 3-8 and described in the following paragraphs.

Table 3-8. Selected Visualization techniques

Technique	Used to Illustrate
Hinton diagram	Weights and biases
Bond diagram	Weights and biases
Hyperplane/hyperplane animator	Hidden units affecting decisions
Trajectory diagram	Weight space
GUI/KBANN	Topology and initial weights
Lascaux	Forward propagations of activations Backward propagation of error Changes to weights and biases

The Hinton diagram, developed in 1986, was one of the first visualization methods. It provides a compact visual display of the weights and biases related to a particular NN [Hinton 1986]. Figure 3-13 depicts an NN and the Hinton diagram to visualize the network.

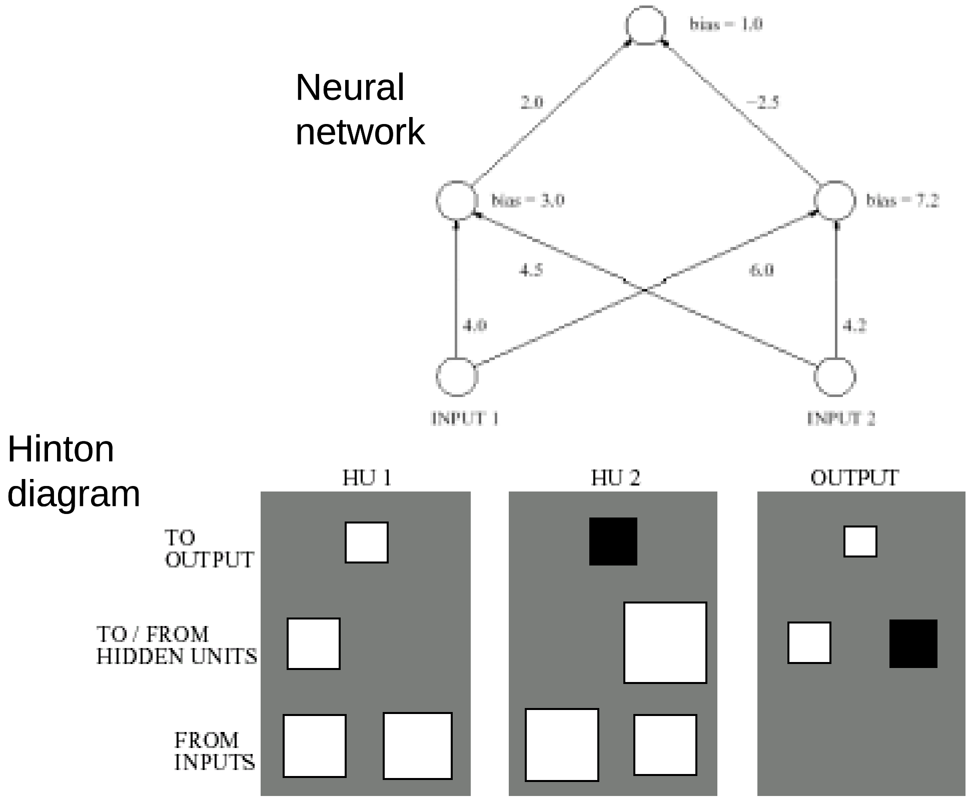


Figure 3-13. Neural Network and Related Hinton Diagram

These diagrams show the two hidden units and the output unit of the network. The boxes in the lower part of each diagram depict weights from (to) hidden units, and the boxes in the middle of each diagram depict a weight to the output unit. A unit's bias is drawn in the position in the unit's diagram where weights to and from the unit are shown in the other diagrams. The Hinton diagram is a rather weak method for visualization because the topology is not readily apparent from the diagram and it does not clearly show how a unit partitions its input space.

Wejchert and Tesauro developed the bond diagram in 1990 [Wejchert 1990]. This visualization method illustrates the sign and magnitude of each weight and bias in the network, but, unlike the Hinton diagram, it does show the topology of the NN. In the bond diagram, each unit is represented as a disk. The size of the disk indicates the magnitude of the unit's bias. The weights are represented by the bonds linking the disks. The amount (width) of the bond indicates the magnitude of the weight, and the color represents the sign. Figure 3-14 shows a bond diagram for the simple NN structure presented in Figure 3-13.

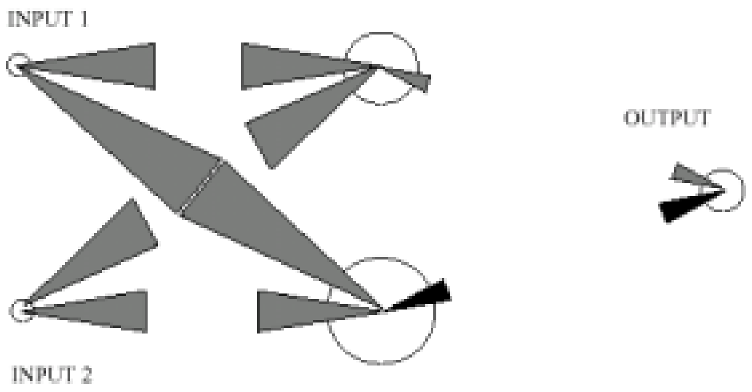


Figure 3-14. Bond Diagram

Since different geometric forms are used to depict weights and biases, it is not as easy to see how the weights compare to the biases. This is not the case in the Hinton diagram, where both weights and biases are boxes. Using the same geometric object can provide visual information on the relative magnitudes of the weights versus the biases. This can help answer a question such as “Which input units need to be active in order for the net input to exceed the bias of this hidden unit?”

One way to visualize the learning process is to graphically display the movement of the hyperplane in the input space of the unit that the hyperplane represents [Munro 1991; Pratt 1991]. A hyperplane diagram can show how hidden units make decisions in an input space defined by input units, or it can show how output units make decisions in an input space defined by hidden units. Figure 3-15 shows the hyperplane diagram of the NN pictured in Figure 3-13.

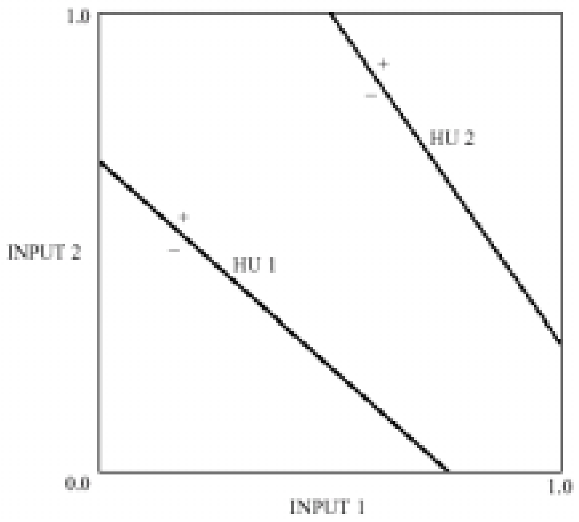
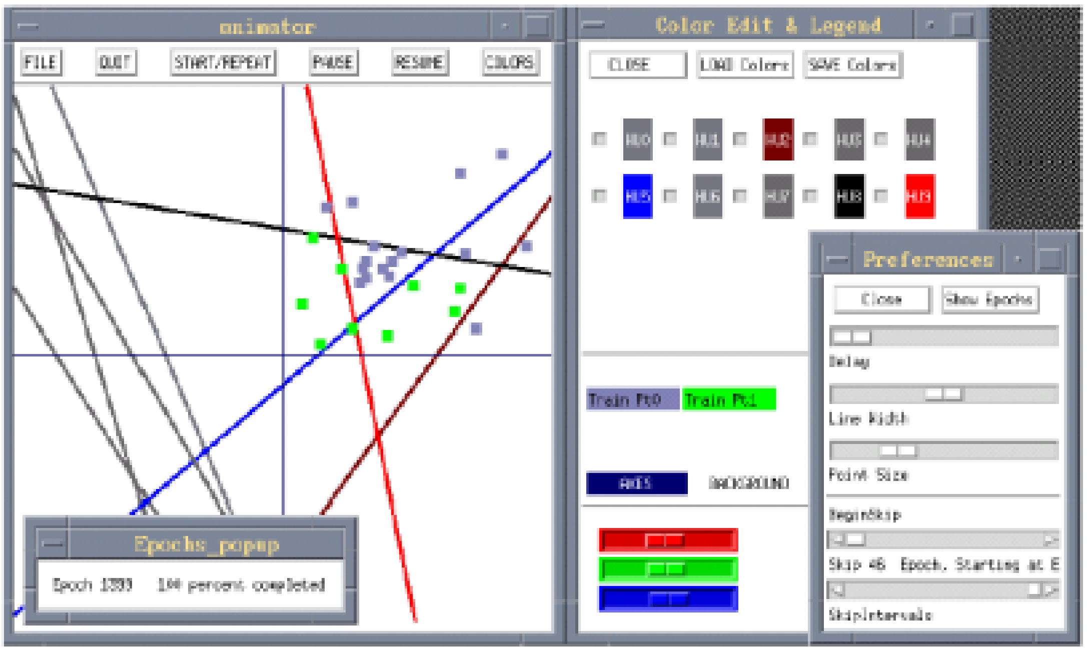


Figure 3-15. Hyperplane Diagram

The axes of the diagram denote the range of activations that may be propagated to the units through their incoming connections. Data points that a network is learning to classify may be plotted in the space. Each hidden unit of the network is represented by the hyperplane (in this case the line) that indicates how the unit is partitioning the input space. The learning process is automated by showing the movement of the hyperplane as the weights and biases of the network are changed.

One limitation of a hyperplane diagram is that only two- or three-dimensional input spaces can be depicted. Selecting a two- or three-dimensional projection of the actual input space may be used to depict an input space of higher dimensionality. There may be a problem choosing which projection to view. Statistical techniques, such as principal component analysis or canonical discriminant analysis, may be useful in determining which projections would provide the most information.

Hyperplane representation can also be animated. Pratt and Nicodemus [Pratt 1993] reported on case studies using a hyperplane animator that they developed, pictured in Figure 3-16. The animator is able to display the relationship between a network and the training data, and is also able to show the changes in that relationship during learning.



Epoch 1399 100 percent completed

Skip 40 Epoch, Starting at E

Figure 3-16. Sample Screen from Hyperplane Animator

The trajectory diagram is another visualization method developed by Wejchert and Tesauro [Wejchert 1990]. The trajectory diagram is designed to provide insight into the weight space for a given problem. A trajectory diagram depicts the movement of a given unit through the weight space. Figure 3-17 shows the trajectory over a hypothetical training session of the rightmost unit in Figure 3-13.

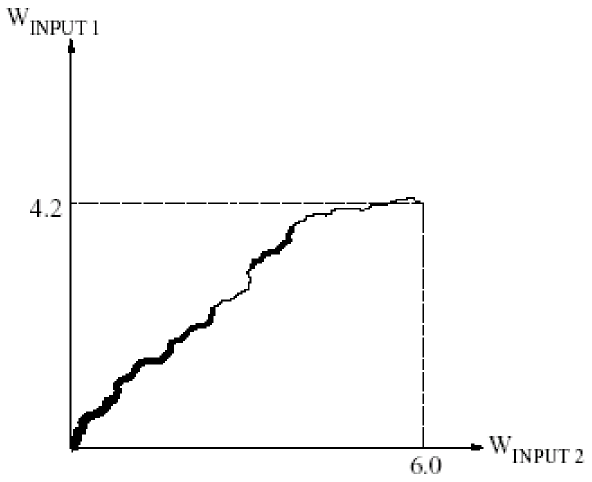
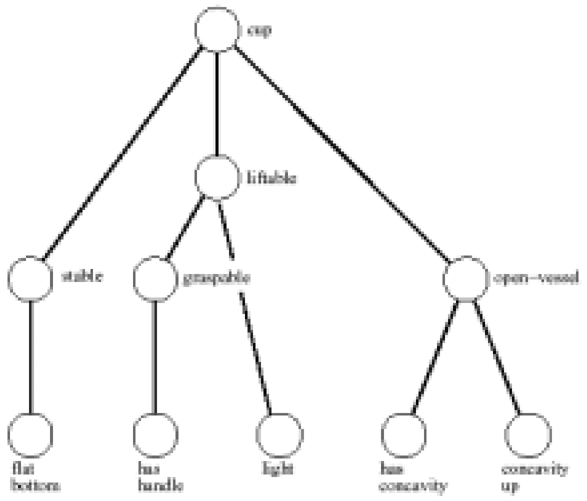


Figure 3-17. Trajectory Diagram

The trajectory is plotted in the space defined by the two weights impinging on this hidden unit. The thickness of the trajectory line indicates the network error along the trajectory. A network unit at a given point in time is plotted as a point in the diagram; the coordinates of the point are specified by the values of the weights feeding into the unit. As learning progresses, the point is replotted to reflect the updated values of its incoming weights

A weakness of the trajectory diagram is the inability to visualize high-dimensional weight spaces. These diagrams have only minimal usefulness because of this limitation. Attempts to visualize higher-dimension weight spaces by projection may lead to diagrams that are not unique.

A graphical interface for visualizing knowledge-based NNs has been developed by the University of Wisconsin. A weakness of conventional NNs is that they provide no way to exploit existing knowledge about the problem to be solved. The knowledge-based NN (KBANN) algorithm [Towell 1990] provides an approach to incorporating existing knowledge into an NN. The KBANN algorithm uses a knowledge base of domain-specific inference rules in the form of PROLOG-like clauses to determine the topology and initial weights of an NN. The domain theory does not need to be complete or correct; it need only support approximately correct domain reasoning. KBANN translates a domain theory into an NN in which units and links correspond to parts of the domain theory. Consider the approximately-correct domain theory for recognizing cups, which is depicted in Figure 3-18.



cup	:-	stable, liftable, open-vessel
stable	:-	flat-bottom
liftable	:-	graspable, light
graspable	:-	has-handle
open-vessel	:-	has-concavity, concavity-up

Figure 3-18. Hierarchical Structure of Cup Domain Theory

The hierarchical structure of the domain determines the topology of the knowledge-based NN: the input units of the network represent the base-level facts of the domain theory, the hidden units represent intermediate conclusions, and the output unit represents the final conclusion. After the network topology and initial weights have been determined by KBANN, the network is trained using the backpropagation algorithm and a set of training examples. After training, refined rules can be extracted from the network [Towell 1991].

Lascaux is another tool developed by the same group at the University of Wisconsin. It assists in further visualizing the NN both during and after learning. This tool enables visualization of the learning process by depicting forward propagation of activations, the backward propagation of error, and changes to the weights and biases of the network. Each network unit is represented by a box that is labeled with the concept represented by that unit. Lines that connect the units represent network weights. The thickness of each line indicates its magnitude, with positive weights drawn as solid lines and negative weights as dashed lines. Figure 3-19 shows the interface provided by the Lascaux tool.

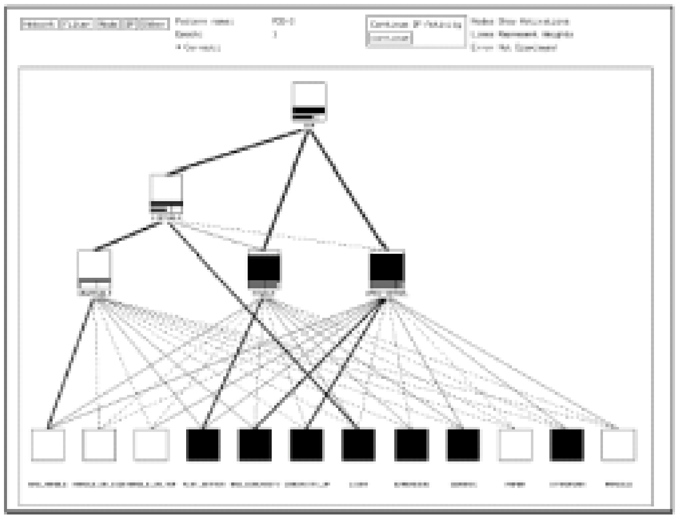


Figure 3-19. Lascaux Depiction of a Knowledge-based Neural Network

The activation of each unit is depicted by a thermometer-like display. The activation meter occupies the top portion of each box and the level to which the meter is filled with black indicates the activation of the corresponding unit. The lower part of the box shows the net input relative to the "threshold" of the unit. In addition to showing the activation of each unit, Lascaux can display the error of each hidden or output unit for a particular pattern of learning.

Lascaux also includes mechanisms for filtering the information that is to be displayed. For example, a user-settable threshold enables the user to view a subset of the weights of the network; weights less than a chosen threshold will not be displayed. Another mechanism allows the user to select units or deselect units for which the weights can be displayed. Mechanisms such as these are important features of a visualization tool to insure that the user will not be overwhelmed by the magnitude of the data available for display.

The interface can also depict the forward propagation signals from unit to unit as shown in Figure 3-20.

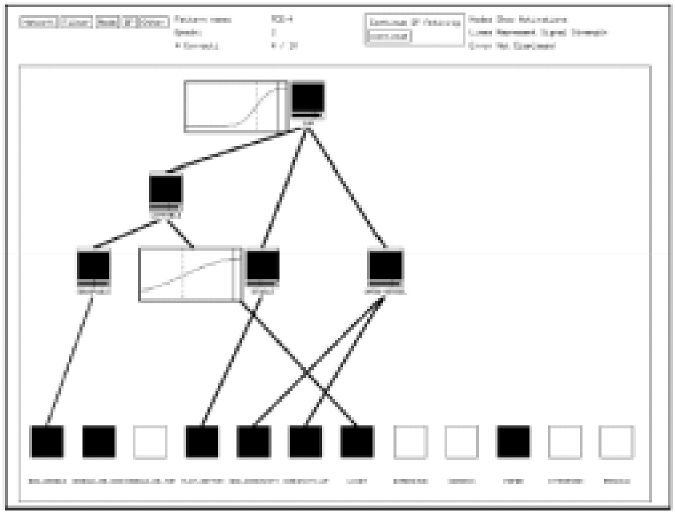


Figure 3-20. Activation signals and effective activation functions

The diagram plots the activation function for a unit on a scale that is defined by the range of the net input values that the unit could have. Thus, the rightmost edge of the diagram shows the activation value that would result if the unit were to receive its maximum net input. The leftmost edge shows the activation that would result if the unit were to receive its minimum net input. The actual net input that results for a given pattern is displayed as a solid vertical line in the diagram. This displays the effective activation. It is valuable to describe the nature of the activation function relative to its weight space and to show the relative influence of the weights and biases. Lascaux also provides a mechanism to specify a "freeze" display that lets the user progress step-by-step through a set of input patterns.

Lascaux provides the same functionality whether it is used with conventional ANNs or with knowledge-based NNs. The tool aids in understanding the refinements that occur during learning by animating the weight changes. This can help explain why the network has made a particular decision.

In summary, visualization techniques can provide insight into the workings of a network by transforming the parameters into more easily understood visual representations.

Although visualizations can help with the understanding of parameters in an NN, the techniques are still problematic and cannot completely address the aspect of the high-dimensionality of the spaces that need to be understood. One challenge of future network visualization work is to develop methods that can succinctly compress these high-dimensional spaces into easily understood and meaningful representations.

3.2.7 Model Checking

For autonomous software, traditional testing methods fall short because the combinatorial explosion of input possibilities results in a set of situations too large to be analyzed. Formal methods of verification become necessary for such software. Through mathematically based analysis, model checking establishes that the program fulfills formally expressed requirements. Formal verification techniques based on model checking are able to efficiently check all possible execution traces of a system in a fully automated way. Given a model of the system and an expected property of the system, a model checker will run through all possible executions of the system and report any execution that leads to a property violation. In the past, manual conversion to the syntax accepted by the model checker was required, which made the use of these tools tedious and complex. This conversion usually required a good knowledge of the model checker, and was usually carried out externally by a formal methods expert instead of the system designer. Translators have been developed [Pecheur 2000] and new model checkers have immerged [Brat 2000] that make model checking a more accessible tool.

Two approaches to model checking are:

- Theorem provers – computer supported proof of the requirements by logical induction over the structure of the program. (Off-line research studies)
- Model checkers – search of all realizable executions of the program for a violation of the requirements. (On-line automatic)

Formal model checking methods applied in the design phase catch errors early and reduce maintenance costs later on. Although this is a crucial early step there are arguments for formal methods to be applied to the programs as well. Programs may contain fatal errors in spite of careful design. Modern programming languages are well developed and a result of good language design principles so they may be good design/modeling languages. Also formal methods can verify program correctness while debugging and locating errors. Turning programs into verifiable models allows for model checking of software. First, the program is translated into the language of the model checker. Then some of the original system must be "abstracted" away to obtain a model that can be checked in a reasonable time and space [Pecheur 2000].

Classical explicit model checkers, such as SPIN, generate and explore every single state of the model. Symbolic model checkers, such as Symbolic Model Verifier (SMV), manipulate whole sets of states at once. Java Path Finder (JPF) is an explicit model checker that employs the use of abstraction, static analysis, and runtime analysis in order to alleviate some of the state-explosion problems. Symbolic model checkers implicitly represent the set of states as the logical conditions that those states must satisfy. These conditions are encoded into data structures called Binary Decision Diagrams (BDD). The BDD of the current set of states is combined with the BDD of the transition relation to obtain the next set of reachable states. The BDDs provide compact representations and support very efficient manipulations. Model checking was traditionally applied to hardware systems but is increasingly being applied to software systems.

In many of the early model checkers, translation from the modeling language, Modeling Programming Language (MPL), to the language of the model checker was done by hand. This conversion was a very complex task that could take weeks or months and that usually needed to be performed by formal methods experts. The running of the verification only took minutes or hours after the translation was accomplished. The complexity and time-consuming nature of the translation led to the development of translators from MPL to the model checker language to automate the process. Pecheur and Simmons developed one such translator to convert MPL to SMV [Pecheur 2000]. The translator was applied in several settings within NASA, such as Livingstone, an MPL, model-based diagnosis and recovery system for the Remote Agent architecture on the Deep Space One spacecraft. Several minor bugs were found even after the models had been extensively tested by more conventional means. The translator has also been used at Kennedy Space Center by the developers of the Livingstone model for the In-Situ Propellant Production (ISPP). The current version of the ISPP model, with 10^{50} states can be processed in less that a minute using SMV.

There are now model-checking tools such as Java PathFinder that can be applied to implementation programs. These tools allow software developers to check their own code during development. Java PathFinder 1 (JPF1) uses automatic translation from Java to PROcess MEta Language (PROMELA) the input language for the SPIN model checker. Java PathFinder 2 (JPF2) is an updated version of JPF1 that handles additional Java language features [Brat 2000]

While model checking does not prove the correctness of a model it is a good compliment to testing because it allows for a wider coverage at a lower cost. Referred to as "falsification method", it can be used to prove systems wrong rather than right. While model checking is a powerful and flexible debugging tool that can be used early on in the engineering process it only addresses validity as an abstraction of a physical system. The most challenging part of model checking programs is reducing the size of the state space to something the tool can handle. These programs can be coupled with abstract interpretation, static analysis, and run-time analysis to handle this problem. Analytical testing can also be employed and is halfway between testing and model checking. This intermediate approach would provide better accuracy of the verification results.

Challenges in model checking include:

- Different kinds of abstractions, discrete, real time, continuous and hybrid model mix.
- Adaptive systems designed to modify behavior dynamically.

Benefits of model checking include:

- Use of the abstract model, rather than the actual code.
- Executes in a highly efficient way using backtracking to explore alternative paths from common intermediate states.
- Can be applied in early stages of design long before testable implementation is available.

Limits of model checking include:

- Limited by state-space explosion. Since induction cannot be performed, only systems of bounded size can be verified.
- The process of abstraction of the model can be time consuming and costly and is usually performed off-track by V&V experts.

3.3 Summary of Tools

The summary of tools is heavily influenced by the summary of methods. Some of the methods contain tools, which have already been developed for method application. The bulk of this summary will contain those tools. A few additional tools discovered during non-specific searches have also been evaluated. As some tools can be classified into two or more of the methods, the tool summary is presented in an alphabetical fashion.

Each tool description will try to evaluate the tool across several different criteria. These criteria include expense, ease of use, translation requirements, automated features, the tool's track record and available support to aid in tool usage. To facilitate quick assessment, Section 4.0 will present each tool's evaluation in an easy to read table.

3.3.1 HyTech

Website: http://www-cad.eecs.berkeley.edu/~tah/HyTech/

HyTech was developed by Tom Henzinger, Pei-Hsin Ho, and Howard Wong-Toi at the University of California at Berkeley. This tool is a symbolic model checker for linear hybrid automata, a subclass of hybrid automata that can be analyzed automatically by computing with polyhedral state sets.

A hybrid system [Henzinger 1997a] is a dynamical system whose behavior exhibits both discrete and continuous change. A hybrid automaton is a mathematical model for hybrid systems, which combines, in a single formalism, automaton transitions for capturing discrete change with differential equations for capturing continuous change.

A key feature of HyTech is its ability to perform parametric analysis, i.e. to determine the values of design parameters for which a linear hybrid automaton satisfies a temporal-logic requirement. In particular, HyTech is an automatic tool for the analysis of embedded systems. It works by computing conditions under which a linear hybrid system satisfies a temporal requirement. The HyTech tool provides not only a model checker but also a parametric analyzer.

The standard reference to the HyTech algorithm is explained in [Alur 1996], and the standard reference to the HyTech tool is provided in [Henzinger 1997b]. Also available is the *HyTech User Guide* [Henzinger 1995], with instructions for installation and usage.

HyTech has been through two major revisions, the final version being written entirely in C++. A linear hybrid automaton description language for specifying the system to be evaluated is provided by the tool. Major language components include: variables, locations, initial conditions, invariant conditions, transitions, and rate conditions.

The analysis of HyTech is based upon symbolic region manipulation techniques first presented for real-time systems in [Henzinger 1994]. The input file consists of a text file containing a system description and a list of iterative analysis commands to be performed.

HyTech is freely available from the HyTech website. However, one does need to sign a license agreement before downloading. Versions exist for UNIX—including Sun OS 5.8, Solaris 2.3.x, Digital UNIX, DEC Ultrix, and HP-UX—Linux, and Windows. The Windows version requires use of the Cygwin package that ports UNIX tools to Windows. The tool developers have provided a user guide to assist in its installation and usage.

Since it was developed for the UNIX environment, HyTech is available as a compressed tar file that needs to be decompressed and expanded – thus generating the appropriate directory tree and files. As a command-line program, it accepts various parameters to redirect I/O and specify the levels of checking and analysis to be performed. However, a compatible graphical input language for HyTech is available courtesy of the UPPAAL group in Denmark and Sweden. UPPAAL is discussed in more detail in Section 3.3.17.

Case studies involving usage of the tool included an audio control protocol [Ho 1995], and a steam-boiler control [Henzinger 1996].

The audio control protocol example demonstrated that:

- HyTech's symbolic model-checking is expressive: it is not limited to systems with discrete state spaces—being able to verify infinite state systems with continuous variables subject to variable rates of change,
- The transmission of arbitrary length messages can be modeled using only finite-state data information, and
- Timing properties involving arbitrarily large timing constants also can be verified.

The modeling of a steam-boiler control system used hybrid automata. Abstracted linear models of the nonlinear behavior of the boiler were defined and verified. In particular, HyTech was able to automatically synthesize control-parameter constraints that guarantee the safety of the boiler.

3.3.2 Java PathExplorer (JPAX)

JPAX is a runtime verification system for monitoring Java execution traces. It combines testing with formal methods to leverage the strengths and offset the downside of both. JPAX can be used during testing, and can potentially be used during operations to check safety critical systems. It extracts state events from the target application as it runs, then analyzes the collection using a remote observer process.

The developers' goal is to make the system as general and generic as possible, handling multiple languages and allowing user-defined verification rules and specification logics. [Havelund 2001]. Eventually, JPAX will also be able to monitor subprograms written in C, C++, and others.

The tool assists in performing two types of verification: logic based monitoring, and error pattern analysis. Together, the developers state, “JPAX offers a large, if not a full, spectrum of possibilities for runtime verification” [Havelund 2001].

Logic based monitoring checks the program against the underlying logic expressed by the user requirements. By using a specification runtime language, a user can create formal requirements specifications that can be compared against recorded execution traces.

The developers of JPAX chose to use Maude, a modularized specification and verification system that implements rewriting logic. Maude offered high-performance and the ability to define new logics including temporal logics such as future time and past time linear logic. As the type of logic used may vary from system to system, defining new logics in Maude would be feasible for advanced users but may be unfeasible for IV&V personnel new to this language. The Maude language is capable of performing 15 million rewritings per second, and can be used as the monitoring engine that performs the conformance checks using Linear Temporal Logic (LTL).

Error pattern analysis uses standard language-dependent algorithms to detect typical concurrency error potentials. Through the use of various algorithms, error pattern analysis can identify error-prone programming practices, particularly those leading to data race conditions and deadlocks. Errors need not occur in the test run in order to be detected in a recorded execution trace of a program. JPAX can evaluate the order and frequency of lock or semaphore access and determine if these events would lead to future conflict.

For now, JPAX is currently designed to operate on the JAVA run-time language. JPAX requires two sets of inputs: the Java bytecode (created with the standard Java compiler) and the specification script defining the analysis. The specification script consists of two parts: an instrumentation script that defines how the program should be instrumented, and a verification script that identifies the exact analysis to be performed and, if logic based monitoring is requested, what properties should be verified. The scripts are written in Java, which initiates Maude as needed; Java defines the Boolean predicates and distributes the values of those to Maude for deeper analysis.

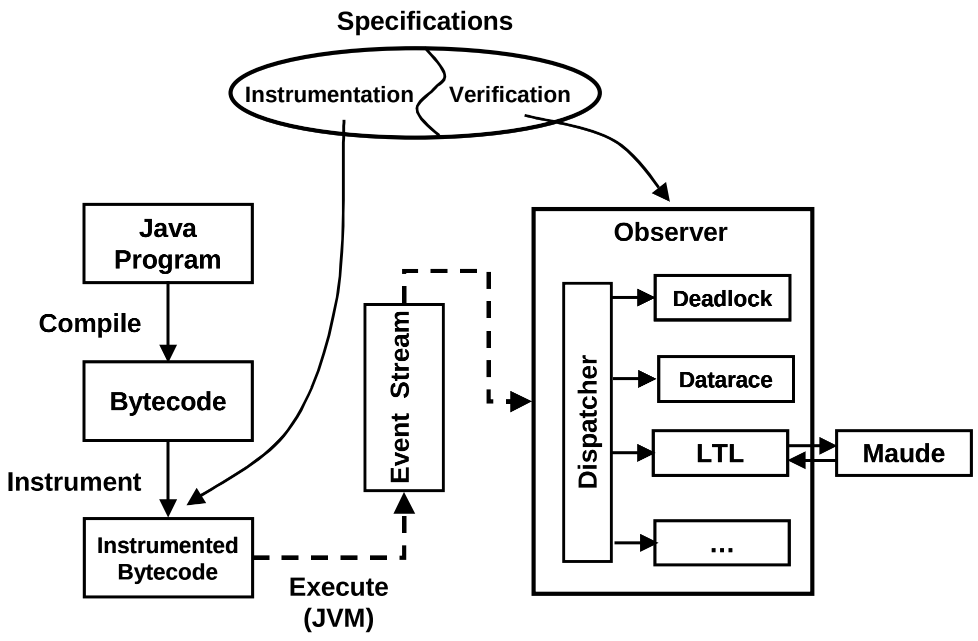


Figure 3-21. JPAX Architecture

There are three modules that make up the JPAX tool. An instrumentation module performs script-specified automated instrumentation of the target program. Relevant results are sent to an interconnection module, which forwards them to the observation module that can perform analysis on the system. Instrumentation is performed using the Compaq bytecode engineering tool Jtrek, which reads Java class bytefiles, examines their contents while traversing them as abstract syntax trees, and inserts new code that can access the contents of a call-time stack at runtime. The contents are transmitted as events to the observer, which dispatches the events to a set of rules, each rule performing a requested analysis, as shown in Figure 3-21. Rules may be written in Maude, Java, or C. The only language-specific portion of the system is the instrumentation module, which can be replaced to set up for a different language.

Like all runtime programs, JPAX slows normal execution of the program. JPAX relies on two buffers for monitoring program execution. One buffer stems from the instrumented program to the observer, and the other from the observer to Maude. The slowdown, the developers believe, comes from the buffer communication between the observer implemented in Java and the logic engine implemented in Maude. As a result, one area of investigation is to devise Java implementations for the most heavily used logics to check formulae directly against traces, thereby eliminating some communication.

Future developments planned for JPAX:

- Investigate more suitable logics for monitoring (than future time LTL)
- Experiment with new logics in Maude more appropriate to monitoring than LTL, such as interval and real time logics and UML notations
- Fast implementations of designated logics in more conventional programming languages than Maude (improving overall monitoring speed)

- Develop new error pattern analysis algorithms to detect concurrency errors beyond data races and deadlocks
- Study new functionalities such as guided execution via code instrumentation to explore more interleavings of a non-deterministic concurrent program during testing.
- Guidance of program during operation once a requirement specification has been violated
- Dynamic programming visualization.
- More user-friendly interface.

3.3.3 Java PathFinder (JPF) and Java PathFinder 2 (JPF2)

The original JPF application was strictly a translator, converting code from a subset of Java 1.0 into the PROMELA modeling language for input into the SPIN model checker. JPF was developed by the Automated Software Engineering (ASE) group at NASA ARC. The initial JPF had numerous limitations: Translation was restricted to features available in destination language (PROMELA), so items, such as floating point numbers, could not be handled. Furthermore, the translation process required the original source code, which might not be readily available. As stated in Section 3.1.1, JPF did prove its merit when it was used successfully with SPIN in locating the errors that caused the Remote Agent to hang [Havelund 2000].

To overcome some of the problems associated with JPF, NASA ARC recreated the tool. The new iteration of Java Pathfinder, JPF2, is an explicit-state model checker developed to work directly on Java bytecode. It is capable of model checking all of Java, not just subsets, and can detect problems that can only be discovered at the bytecode level. Among its strengths is a customized Java Virtual Machine (JVM) implementation tailored for efficient memory management, a critical concern when examining a program with many states.

JPF2 consists of two components: a special-purpose Java Virtual Machine (MC-JVM) tailored for efficient memory management – an important concern since state-explosion can rapidly consume memory – and a depth-first algorithm. The MC-JVM keeps track of states that have been visited by storing all new states in a hash table (index). It also maintains the path of states in a stack to permit navigation. The depth-first algorithm performs the actual traversal of the state-graph of the program. It instructs the MC-JVM to evaluate the current invariant, to move forward one step, or to move back one step.

Novel features of JPF2 cited by [Brat 2000] include the following:

- Canonical heap representation - memory is always allocated to the same location as during previous interleaving
- Garbage collection – to reduce cluttering of memory
- Traps for certain method-calls to return a non-deterministic integer value
- Adjustable atomicity levels (one bytecode instruction, bytecodes for one Java instruction, bytecodes for one line of Java code, or all bytes within a block of Java code that are independent to any concurrent code)
- Highly structured program states consisting of a number of Java classes (rather than traditional, memory-hogging flat, state-vector style of many model checkers).

JPF2 offers the ability to 1) automate abstraction to convert an infinite-state program into a finite one; 2) use runtime analysis to locate problem codes; 3) use static slicing to reduce the program; and 4) launch the model checker to analyze the result.

3.3.3.1 Static Analysis

Static analysis searches the program for "safe" program blocks – that is, standalone blocks of consecutive Java statements that can be executed together because they only use local data. JPF2 uses a static slicing tool from the BANDERA toolset (Kansas State University) to identify program dependencies; the tool automatically extracts slicing criteria for both sequential and concurrent programs. The static analyzer uses this information to identify the safe programming blocks that are used by the MC-JVM of JPF2 to compute "mega" steps (program slices). These mega steps can be safely executed in parallel to provide partial order reduction and reduce state explosion.

3.3.3.2 Runtime Analysis

Runtime checkers employ algorithms to predict execution traces of the target program that may violate properties of interest. One of the algorithms used by JPF2 is Eraser, used to dynamically detect data race conditions in multi-threaded programs. Eraser works by searching for the absence of locks or semaphores, flags used to indicate that a single piece of code has sole permission to access some data. A failure could occur should two pieces of code access data at the same time with one code segment writing the data while the other reads it. A program would be considered race free if, "for every variable there is a nonempty set of locks that all threads own when they access the variable. [Brat 2000]" Another run-time algorithm used analyzes the lock order – checking whether a lock or semaphore can be taken in different orders by different threads (which can lead to deadlocks from threads that are waiting for each other to give up their respective lock before they can continue).

While algorithms like Eraser can provide the information to identify the error, the model checker is the key in analyzing the consequences. Once a race condition is noted, then JPF2's special runtime analysis/model checking mode launches a window showing the threads involved in the race condition. The model checker can then be used to see what happens when one thread is chosen, with priority over another thread, to analyze the race condition.

3.3.3.3 Automated Abstraction

An automated abstraction tool that can be used to reduce an infinitely large number of states to a smaller set also makes up part of JPF2. The user may specify (Boolean) variables to be removed and/or added, and JPF2 will automatically generate a new Java program using the new abstract variables and unremoved variables. In this way, the user can abstract subcomponents of a program too large or complex to abstract in total, or can create new variables that depend on variables from multiple classes (interclass abstraction). In the automatic conversion, the tool uses Stanford Validity Checker (SVC) to check the validity of the logical expressions.

3.3.3.4 Usage and Development

While JPF was used in reviewing the Remote Agent program for errors that caused in-flight deadlock problems, no literature on the use of its successor, JPF2, appears to exist. This may not be the case in the future though; ASE is collaborating with the Advanced Architectures and Agents Group at Goddard Space Flight Center and Global Science and Technology to apply JPF2 to analyze satellite downlink protocol [Pecheur 2000].

In 2000, the JPF2 development team was in the final stages of creating a parallel version that runs on multiple workstations. Other improvements will include extending the program to do LTL model checking and developing a more flexible specification for Java [Brat 2000].

3.3.4 KRONOS

Website: http://www-verimag.imag.fr/PEOPLE/Sergio.Yovine/kronos/index.html

KRONOS is a verification tool for real-time systems, developed at VERIMAG (http://www-verimag.imag.fr/), an academic laboratory focusing on the theoretical and practical aspects of formal methods for software engineering.

KRONOS was developed for the verification of complex real-time systems. Such systems are often part of complex safety-critical applications such as aircraft avionics. While these high assurance systems are very difficult to design and analyze, their correct behavior must be ensured because failures may result in severe consequences. The goal of KRONOS is to formally prove their correctness with respect to the desired requirements.

In KRONOS, components of real-time systems are modeled by timed automata. Timed automata are automata extended with a finite set of real-valued clocks, used to express timing constraints. The correctness requirements are expressed in the real-time timed computation tree logic (TCTL). TCTL, proposed by Alur in 1991 [Henzinger 1992], is a timed extension of the well-known temporal logic called Computation Tree Logic (CTL) which was itself proposed by Clarke and Emerson in 1981. TCTL allows quantitative temporal reasoning over dense time.

KRONOS checks whether a timed automaton satisfies a TCTL-formula. The model-checking algorithm is based upon a symbolic representation of the infinite state space by sets of linear constraints.

Timed automata are finite-state machines equipped with a set of variables, called clocks, that measure the elapsed time between events. The automaton models the behavior of a process or component of the system. The locations of the automaton correspond to various points in the process. Finally, transitions between locations correspond to the execution of statements.

Clocks can be set to zero and their values increase uniformly with time. At any instant the value of a clock is equal to the time elapsed since the last time it was reset. A transition is enabled only if the timing constraint associated with it is satisfied by the current values of the clocks.

A system in KRONOS is specified as a set of files, say com-1.tg, com-2.tg, …, com-n.tg with each specifying the behavior of one component. The major features of a system's behavior are global states and global evolutions.

- Global states—a global control location is an m-tuple of locations of the system's components. At a given moment in time, the global state is determined by this control location and the values of the clocks of all the components.
- Global evolutions—the system changes between global states by allowing time to elapse, or by execution of a transition.

- Time passing—all components must agree in the amount of time that can elapse.
- Execution of transitions—the global location of the system changes when a single component executes a transition, or if a subset of components executes synchronized transitions.
- Product automaton—the essence of the system's behavior is described by a single timed automaton.

KRONOS provides a specification framework that integrates both logical and behavioral approaches to verification. The correctness criteria may be specified using either approach.

For the logical approach, formulas of the timed temporal logic are used with model-checking algorithms to check whether the systems satisfies a property represented by a TCTL formula.

The logical approach facilitates the analysis of various classes of properties. Examples of properties amenable to the logical approach include reachability, non-Zenoness, and bounded response. Reachability, an example of a safety property, is used to verify that a system never can achieve an unsafe state. Bounded response describes the real-time character of the system: "Can events from the system's environment be responded to in a bounded time interval?" Non-Zenoness concerns the divergence of time: "Can the system reach a state where time converges, called Zeno, effectively stopping the flow of time?" Satisfying safety criteria by stopping the flow of time is clearly unacceptable.

For the behavioral approach, KRONOS provides an algorithm that constructs an automaton in which time is abstracted away but the causality relationship is preserved. This facilitates the generation of behavioral equivalences, which have proven quite useful in the verification of concurrent systems.

KRONOS has been under development since 1994. The last version of the product (version 2.2b) was released in 1998. KRONOS was developed for a SUN Solaris 2.5 environment. KRONOS is distributed free of charge to academic institutions for non-profit use. A user guide for KRONOS is available.

KRONOS has been used successfully on real-time systems modeled in several process description formalisms, including: ATP [Nicollin 1992], AORTA [Bradley 1995], ET-LOTUS [Daws 1995], and T-ARGOS [Jourdan 1993].

3.3.5 LOTOS

The Language Of Temporal Ordering Specification (LOTOS) is a formal description technique developed at Twente University in the Netherlands. The International Organization for Standardization (ISO) recognizes it as ISO/IEC 8807.

LOTOS is a language with a high level of abstraction and a strong mathematical basis. It is used for the description and analysis of non-deterministic, complex systems. It consists of two complementary and independent parts:

- The data part is based on the Act One specification and is used to model data structures and value expressions (http://www.run.montefiore.ulg.ac.be/Research/index.php?topic=Lotos).
- The control part is based on a combination of Communicating Sequential Processes (CSP) and Calculus of Communicating Systems (CCS) [Pecheur 1997] and is used to

model dynamic behaviors of systems http://www.run.montefiore.ulg.ac.be/Research/Topics/index.php?topic=Lotos)

The University of Sterling undertook the Specification and Prototyping with LOTOS for an Interactive Customer Environment (SPLICE) project in the early 1990s. One aspect of that project examined how LOTOS could be used to specify a trained perceptron NN. If it could describe the network behavior without the need for execution, it could not only improve customer understanding of the network, but also be a tool for automatic system documentation [Gibson 1993].

A number of tools exist to support specification, verification and code generation using LOTOS (http://www.run.montefiore.ulg.ac.be/Research/Topics/index.php?topic=Lotos). Some popular tools are Cæsar Aldébaran Development Package (CADP), TRAIAN, Graphical LOTOS Animator (GLA), Graphical LOTOS Designer (GLD), Toolset for Product Realization with LOTOS (TOPO), SyMbolic Interactive Lotos Execution (SMILE), and European/Canadian LOTOS Protocol Tool Set (EUCALYPTUS). However, an introductory view of the language is obtained from a table of the main LOTOS operators presented by Pecheur [Pecheur 1997] in Table 3-9.

Table 3-9. Main LOTOS operators

Representation	Explanation
stop	An interactive behavior (like 0 in arithmetic).
G !V ?X:S; B	Interact on gate G, sending V and receiving a value of sort S in X, and then behave as B (other input/output combinations are possible).
B1 [] B2	Behave as either B1 or B2, whichever does something first.
[E] -> B	If E is true then behave as B.
B1 |[G1,…,Gn]| B2	B1 in parallel with B2, synchronized on gates G1,…,Gn (||| means no synchronization, || means full synchronization).
hide G1,…,Gn in B	Make actions of B on gates G1,…,Gn invisible from outside.
exit	Successful termination.
B1 >> B2	B1 followed by B2, when B1 terminates successfully.
B1 [> B2	Behave as B1 until either B1 terminates or B2 performs its first action; in the latter case B1 is discarded.
P [G1,…,Gn] (V1,…,Vm)	Call process P with gate and value parameters G1,…,Gn and V1,…,Vm.

Since most LOTOS users employ a tool, the computational requirements and setup for LOTOS is tool dependent. Consequently, each tool will vary in its ease of use, cost, support, and automation. However, support and background on LOTOS itself is available through many Internet websites and user discussion groups, such as http://www.cs.stir.ac.uk/~kjt/research/well/.

3.3.6 MATLAB NN Toolbox

The MATLAB NN toolbox is an additional software package for MATLAB that provides functions, utilities, and help for creating and training NNs. Its usefulness in regard to the verification and validation of NNs lies with simulation and visualization.

The tool allows for construction of most types of NNs and even provides special utilities for back-propagation networks, radial basis functions, SOMs, and recurrent networks. Once in a model within MATLAB/Simulink, the network can be trained, tested, simulated, and studied. Since MATLAB was designed as a mathematical analysis tool in general, data anywhere in a network is easily accessible for viewing and further manipulation. This means the data is available for analysis utilities like interpolation, statistical analysis, equation solvers, optimization routines, and any of the other powerful MATLAB functions. A system analyst can plot the training error function, watch the change in the weight matrix, and get real-time network outputs to verify their correctness.

The toolbox can be used in both MATLAB and Simulink.

With MATLAB, a user has a C like environment in which to code an NN model. A tester can make use of provided NN toolbox functions, or create their own based upon their specific application. Data to train a network can be imported from several different file formats and the results, like the network training error, can be easily plotted to the screen or saved for later use.

Simulink differs from MATLAB in that it provides a graphical means of programming by making use of blocks, connectors, and menus to control how models behave within the Simulink.

Users can choose from a wide variety of components for use in graphical design. The default Simulink blocks offer 12 different transfer functions, but users can create their own or add in blocks provided by others. Some of the possible transfer functions include hard-limit, linear, and Gaussian activations. A user constructing a graphical simulation also has blocks representing the neuron input functions (such as summation or dot product) and basic functions for application of network weights. An example of the construction of a two-layered, single-input, single-output, feed-forward back-propagation network in MATLAB is shown in Figure 3-22.

Using the Simulink aspect of the toolbox is fairly straightforward; a programmer simply drags and drops desired components, and then connects the different Simulink blocks to create the model. From within the Simulink menu, simulation parameters such as timing, control over data input and data output, and diagnostics can be configured. Just as with MATLAB, Simulink can display results from anywhere within the model to the screen as data plots. Furthermore, networks built in MATLAB are easily converted to Simulink via MATLAB functions (gensim, for example). Figure 3-23 shows the network in Figure 3-22 converted to Simulink.

```
>> net = newff([0 1],[5 1]);
>> gensim(net);
>> net
net =

    Neural Network object:

    architecture:

         numInputs: 1
         numLayers: 2
       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]
     targetConnect: [0 1]

        numOutputs: 1  (read-only)
        numTargets: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

            inputs: {1x1 cell} of inputs
            layers: {2x1 cell} of layers
           outputs: {1x2 cell} containing 1 output
           targets: {1x2 cell} containing 1 target
            biases: {2x1 cell} containing 2 biases
      inputWeights: {2x1 cell} containing 1 input weight
      layerWeights: {2x2 cell} containing 1 layer weight

    functions:

          adaptFcn: 'trains'
           initFcn: 'initlay'
        performFcn: 'mse'
          trainFcn: 'trainlm'

    parameters:

        adaptParam: .passes
         initParam: (none)
      performParam: (none)
        trainParam: .epochs, .goal, .max_fail, .mem_reduc, .min_grad,
                    .min_grad, .mu, .mu_dec, .mu_inc, .show, .time

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors
```

Figure 3-22. Example NN in MATLAB

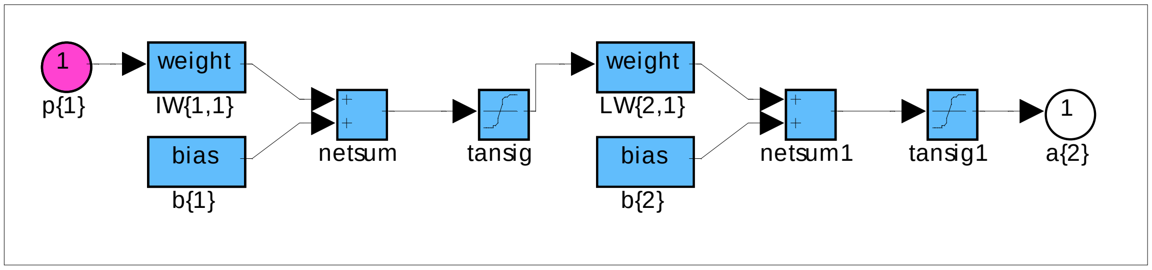


Figure 3-23. Example NN in Simulink

The NN toolbox installs directly into MATLAB/Simulink. It's available for many of the common operating systems, such as all versions of Windows, Linux, UNIX, and Macintosh.

While the NN toolbox requires a MATLAB license, it has powerful features only available with an existing Simulink license. System testers may appreciate the ease of drawing a NN model to simulate and study a particular system as opposed to importing or re-coding the network into MATLAB.

Mathworks, the developer of MATLAB and Simulink, provides a manual for use with the tool, which includes an introduction to NNs and how they can be used. Anyone familiar with the C programming language should find working in MATLAB m-file scripts straightforward. The conventions that MATLAB uses are very close to C and for situations where a system tester wants to do a direct translation of a NN programmed in C, MATLAB offers a C import utility.

The toolbox has been used across a wide range of applications. For the IFC program, the ISR has placed all NNs inside MATLAB m-files and Simulink models.

Mathworks provides good technical support and because of the wide usage of MATLAB, there are several websites from users who have developed their own extensions to this tool and offer their own user stories.

3.3.7 Murphi – or Murφ

Website: http://sprout.stanford.edu/dill/murphi.html

The Murphi description language is based on a collection of guarded commands (condition/action rules), which are executed repeatedly in an infinite loop. This approach is borrowed from J. Misra and K.M. Chandy's Unity model [Misra 1988].

The data structures and guarded commands include well-known data types: subranges, enumerated types, arrays, and records, as well as some new types for improving the efficiency of verification. The new data types include the Multiset, for describing a bounded set of values whose order is irrelevant, and the Scalarset for describing a subrange whose elements can be freely permuted.

Murphi also provides a formal verifier based on explicit state enumeration. The verifier performs depth- or breadth-first searches in the state graph defined by a Murphi description, storing all the states it encounters in a large hash table. When a state is generated that is already in the hash table, the search algorithm does not expand its successor states—as they previously were expanded when the state was originally inserted in the table.

Source code, some executables, documentation, and extensive verification examples are available for Murphi—all as freeware, with very liberal licensing terms. Murphi runs on various versions of UNIX and Linux. The compressed tar file is available at the Murphi Website.

Murphi has been used for the following verification tasks:

- Verification of the cache coherence protocols in Stanford's DASH and FLASH multiprocessors.
- Verification of link-level protocol and cache coherence protocol in Sun's S3.mp multiprocessor.
- Verification of the cache coherence algorithm in Sun's UltraSparc-1.
- Executable specification, analyzer, and verifier for Sparc V9 memory models: TSO, PSO, and RMO.
- Incorporated into University of Wisconsin's Tempest customizable cache coherence protocol system.
- Verification of part of SCI ("Scalable Coherent Interface"), IEEE Std 1596-1992. Some bugs were discovered.
- Analysis of cryptographic and security-related protocols.
- Verification of proprietary protocols at several companies, including Fujitsu, HAL Computer Systems, HP, and IBM.

Cmurphi is another version of Murphi that optionally uses state space caching to speed its performance. Its compressed tar file also is available at the Murphi website.

Parallel Murph, a parallel version of Murphi also has been developed for distributed memory multiprocessors and networks of workstations. In experiments with three complex cache coherence protocols, the parallel Murphi showed close to linear speedups. Since the state table is partitioned over the parallel machine, the algorithm also allows the verification of larger protocols.

3.3.8 PARAGON

Process-algebraic Analysis of Real-time Applications with Graphics-Oriented Notation (PARAGON) is a toolset developed by the Real-Time Systems Group at the University of Pennsylvania for visual specification and verification of distributed real-time systems. It contains:

- A graphical editor for the Graphical Communicating Shared Resources (GCSR) specification language.
- A visual simulator for GCSR specifications.
- XVERSA, a graphical user interface to Verification, Execution and Rewrite System for ACSR (VERSA), a verification tool for Algebra of Communicating Shared Resources (ACSR) specifications.
- Translators between GCSR and ACSR.

Figure 3-24 shows the overall structure of the PARAGON system. The VERSA system provides the verification back-end. It offers state exploration capabilities and checking equivalence of two specifications. XVERSA provides a graphical user interface to the analysis functions of VERSA. GCSR was designed to make the specification language used in both VERSA and XVERSA easy to understand and maintain.

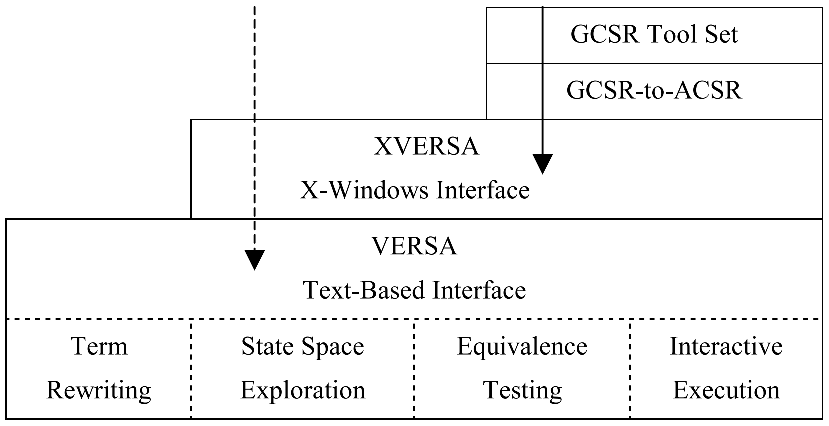


Figure 3-24. The PARAGON Toolset

PARAGON has several extensions associated with it that are designed to interoperate with PARAGON, or use compatible languages and share "look-and-feel." These extensions include:

- Probabilistic ACSR (PACSR), a formalism that associates a probability of failure with every resource.
- The LCSR tool is a model checker based on a graphical version of modal μ-calculus extended with resource modalities. LCSR uses ACSR specifications as input and can be used as an alternative back-end verification tool for PARAGON (http://www.cis.upenn.edu/~lee/paragon.html).

PARAGON is implemented in C++ and X/Motif, with the help of Lex and Yacc compiler construction tools and the LEDA class library to enhance portability. All major components: editor, simulator and the analysis toolkit are implemented as separate UNIX processes and can be used as stand-alone tools (http://www.cis.upenn.edu/~lee/paragon.html).

Information regarding PARAGON was published in the mid to late 1990s; however, its lack of use made the availability of this information sparse. Interested parties were encouraged to obtain a free version of the toolset and information by contacting the authors Lee or Sokolsky.

3.3.9 PAX

PAX is a tool designed for the verification of parameterized systems. In particular, PAX allows one to verify parameterized networks of finite state processes.

The underlying method of PAX is based on three main ideas. First, PAX can model an infinite family of networks by a single WS1S (Weak Second-order theory of 1 successor) transition system whose variables are set (2nd-order) variables and whose transitions are described in WS1S [Buchi 1960]. Second, PAX presents methods that allow one to abstract a WS1S system into a finite state system that can be model-checked. Third, to verify *liveliness* properties, PAX enriches the abstract system with strong *fairness conditions* while preserving safety of the abstraction.

PAX is an integration meta-tool that performs its tasks in the verification process by linking with other V&V tools:

- MONA (http://www.brics.dk/~mona/) - decides WS1S formula and generates automata for satisfying and falsifying interpretations
- SPIN (http://www.netlib.no/netlib/spin/whatispin.html) - serves as a LTL model checker
- SMV (http://www.cs.cmu.edu/~modelcheck/) - checks safety properties
- NuSMV (http://afrodite.itc.it:1024/~nusmv/) - also can be used as a LTL model checker
- Graphviz (http://www.research.att.com/~north/graphviz/) - supports visualization of abstract state graphs

PAX is a text-file based system in which the following files are created with a plain text editor. The sys.init file contains a formula (in correct MONA syntax) characterizing the initial states of the system. The sys.trans file contains the list of concrete transitions (as MONA formula over the concrete pre- and post-variables) together with a unique name. The sys.sdesc contains the basic skeleton of the system. The specification may include macros and predicates, together with some common parts of the transitions. In particular, four comment lines must exist, giving positions on which the abstraction relations, the initial states, and transition formulae can be inserted.

The PAX input files are processed by PAX to create an input file to MONA that describes one abstract transition. The abstract pre- and post-variables occur free, such that MONA constructs an automaton accepting the satisfying interpretations. PAX then uses that automaton to compute the abstract transitions.

The use of PAX involves the following steps (where *sys* stands for the name of the system to being verified):

- Create the input files sys.init, sys.sdesc, sys.trans, and sys.pvars. They specify the system and the abstraction relation.
- Compute the abstract system, comp_abstract_trans sys.
- Make a reachability analysis, ss++ sys, whereas as2spin sys, resp. as2smv sys, constructs a PROMELA, resp. SMV, program for the abstract system.

Verification consists of different steps, dependent on the type of abstraction that is used.

- For proving safety properties, it is sufficient to run the state space exploration after construction of the abstract system. When the safety property is part of the abstraction, it should evaluate in all abstract states to true. ss++ returns a file sys.PAX-states where all reachable abstract states are listed. If there is a state falsifying the safety variable, the parameterized system does not satisfy the property or the abstraction is too coarse.

- For proving individual properties, the program as2spin (or as2smv) can be used to transform the abstract transitions and the initial condition into a PROMELA (or SMV) program. Methods can be used to add fairness requirements to the abstract system that are guaranteed to hold in the concrete system. Then SPIN (or NuSMV) can then be used to prove the property of interest.

PAX has been applied to both asynchronous and synchronous algorithms.

- Asynchronous algorithms
    - Szymanski's mutual exclusion algorithm: Safety
    - Szymanski's mutual exclusion algorithm: Liveness
    - Pnueli's modified version of Szymanski's algorithm
    - Simplification of Dijkstra's mutual exclusion algorithm
    - Dijkstra's mutual exclusion algorithm
    - Cache Coherence Protocols
- Synchronous algorithms (time-triggered)
    - Simple Fault Detection
    - Group Membership Protocol

The PAX Web site, located at http://www.informatik.uni-kiel.de/~kba/pax/, provides links to papers and documentation for PAX.

3.3.10 Planview/Comview

Reid Simmons and Gregory Whelan developed two software validation tools, Comview and Planview, to facilitate human problem solving by using graphics and color to present a "gestalt" view of system execution and interactive facilities for browsing, searching, and tracking down potential problems [Simmons 1997]. Both tools operate by parsing log files and provide examples of system runtime monitors.

Typically, autonomous software, such as RA, uses a concurrent, distributed software architecture that coordinates actions and exchanges information using message passing. Validating and debugging such software can be extremely difficult because subtle flaws in modular interface design may manifest themselves only during system integration. A critical validation test is whether the system responds to stimuli appropriately and in a timely fashion. Faulty algorithms or bottlenecks are two areas of concern.

The two tools parse log files produced during system execution and present data in "intuitively understandable formats." Comview displays patterns of interprocess communication and helps to identify when messages are sent, to whom, and where bottlenecks occur. Planview visualizes execution plans (command sequences) and propagates temporal constraints between plan segments to detect violations that signal potential plan failures. Both are implemented with the Tcl/Tk library. The log file parser is written using lex and yacc. A small portion of the tools is written directly in C.

Comview lays out information in a Gantt chart format, where each module (process or task) is displayed on a separate row, and each rectangle on a row indicates a message received by that module. This is shown in Figure 3-25. The width of the rectangle represents the relative time spent by the module in handling the message. The color of the rectangle indicates whether the module is sending or receiving messages, or waiting for a response. The thin orange bars above the message rectangles indicate when the messages are queued. The layout makes it easy to spot modules that are over/under utilized, where bottlenecks occur, and, if there are regular patterns to the messages, the anomalies stand out.

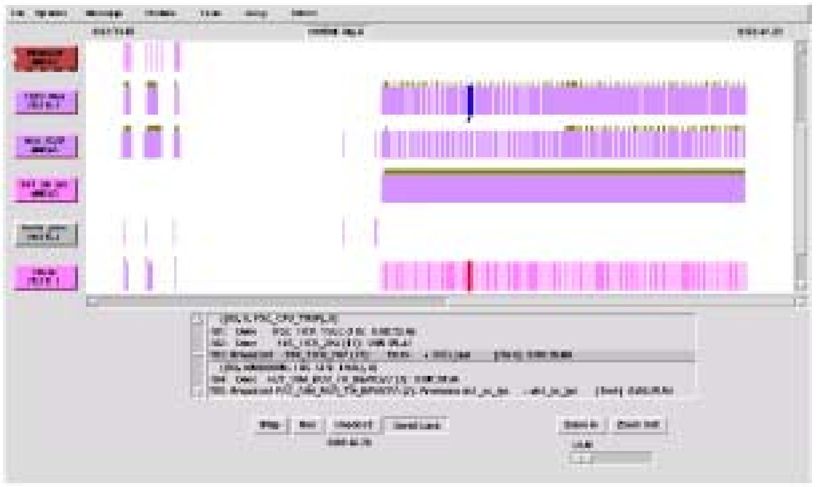


Figure 3-25. The Comview Tool

The tool also provides a hierarchical view. When a module receives a message, it often sends additional messages in response, which are handled and trigger other messages to be sent. This trail is presented as an invocation tree (Figure 3-26), which shows message flow patterns in a visual representation and allows deviations to be spotted.

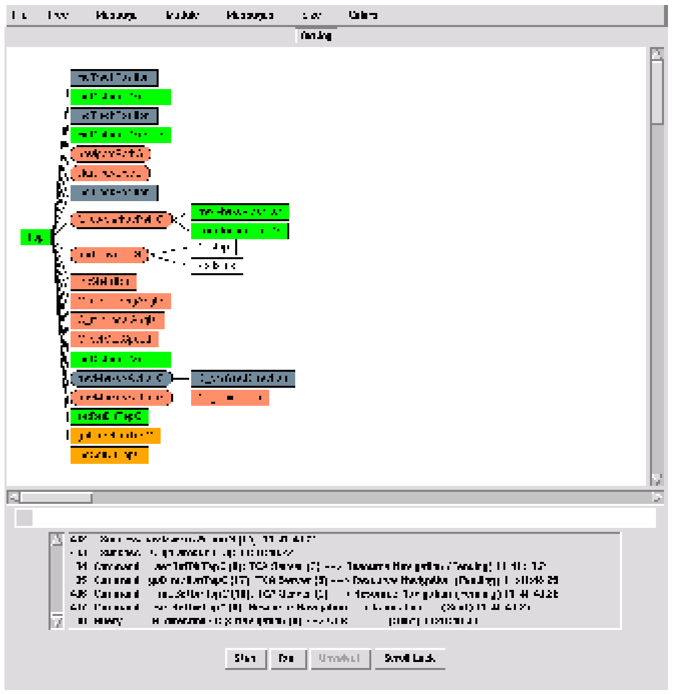


Figure 3-26. A Hierarchical View

If a problem area has been identified, Comview offers features to interactively examine message flow. When a message rectangle is selected from the screen, the communication flow is displayed graphically with an arrow pointing from sender to receiver, and the corresponding line in the log file is highlighted. If the message is queued, the screen displays when it was first sent and when it was eventually handled. Messages can be searched for name, source or destination module, time of message, or content of the message data. Additional display features include rearranging rows, ignoring display of message subsets, zoom and scroll, and change of mappings between message status and colors.

Most autonomous systems have a three-layer architecture:

- A behavioral/real-time layer.
- An executive/sequencing layer.
- A planning layer.

One important aspect in validating an executive is demonstrating that it executes plan segments at the appropriate time and in the appropriate sequence.

Planview was developed to provide that capability for the RA. This executive layer uses a plan representation based on timelines and tokens. A timeline represents the evolution of a state variable (for example, the state of the main engine) over time. The timeline consists of a contiguous sequence of tokens, which represent the value of the state variable over some time interval (for example, the main engine is thrusting).

Tokens have expected duration, start and end times. They may have temporal constraints between them (for example, Token 1 must end before Token 2 begins). The RA executive's task is to achieve the state values associated with the tokens while respecting their temporal constraints and time windows. Faults occur when the executive cannot achieve a token within its specified temporal window. Planview detects violated constraints and helps to track down root causes.

Each row in the Planview display represents one timeline, and each rectangle is a token (Figure 3-27). The position and size of the token indicates when it started and ended (or when it is expected to start and end, if it is in the future). The color represents its status (active, completed, violated, etc.) By selecting a token, textual information is presented in separate windows. Selecting a temporal constraint in the text window causes the constraint to be displayed in the graphical window.

Figure 3-27. The Planview Tool

As Planview processes the log file, token rectangles change in color, size, and location. It automatically propagates the temporal constraints through the token structure, thereby detecting faults such as when a constraint is violated or start or end times are projected to fall outside the respective window. Faults are flagged by changing the color of the affected tokens and by highlighting the affected constraints and/or time windows in the textual display.

The interactive browsing of Planview allows users to follow the chain of constraints manually. The developers also created a facility that automatically generates English language explanations of constraint violations. By selecting a line in the explanation window, the constraint and tokens are highlighted in the graphical display. Planview generates an explanation by annotating which constraints were used for propagation, and then uses those annotations to form a tree of dependencies between tokens. The process is dynamic, so new information and constraints result in updated explanations.

For portability, maintainability, and ease of development, the tools are implemented using standard software packages. The tools are automated using the interprocess communications package to log message traffic. Data that is logged includes indication of source and destination modules, when the message is sent, the data sent with the message, how long the message was queued, and how long the receiving module took to handle the message.

Comview was applied to the Deep Space One software system to identify an unstable control loop, which was caused by a module making decisions based on old data. The software was also used to detect a delay in propagating current data. A certain module was keeping an internal queue, and it was not flushing the queue, but rather sending out only the first element of the cycle. Comview was used to document times when the assumptions were violated.

The Planview tool was designed to provide the capability for demonstrating that the New Millennium RAX executes its plan segments at appropriate times and in the appropriate sequence.

The Planview tool is no longer in use according to Reid Simmons, but the Comview tool is available as part of the InterProcess Communication (IPC) distribution. IPC provides high-level support for connecting processes using TCP/IP sockets and sending data between processes. It takes care of opening sockets, registering messages, and sending and receiving messages, including both anonymous publish/subscribe and client/server type messages. The IPC library contains functions to (1) marshal (serialize) and unmarshal (de-serialize) data, (2) handle data transfer between machines with different Endian conventions, (3) invoke user-defined handlers when a message is received, and (4) invoke user-defined callbacks at set intervals. IPC uses the more efficient UNIX sockets when processes are on the same processor and does byte swapping only when necessary. IPC now supports multi-threaded applications (although currently, it has only been tested under Linux).

3.3.11 PVS

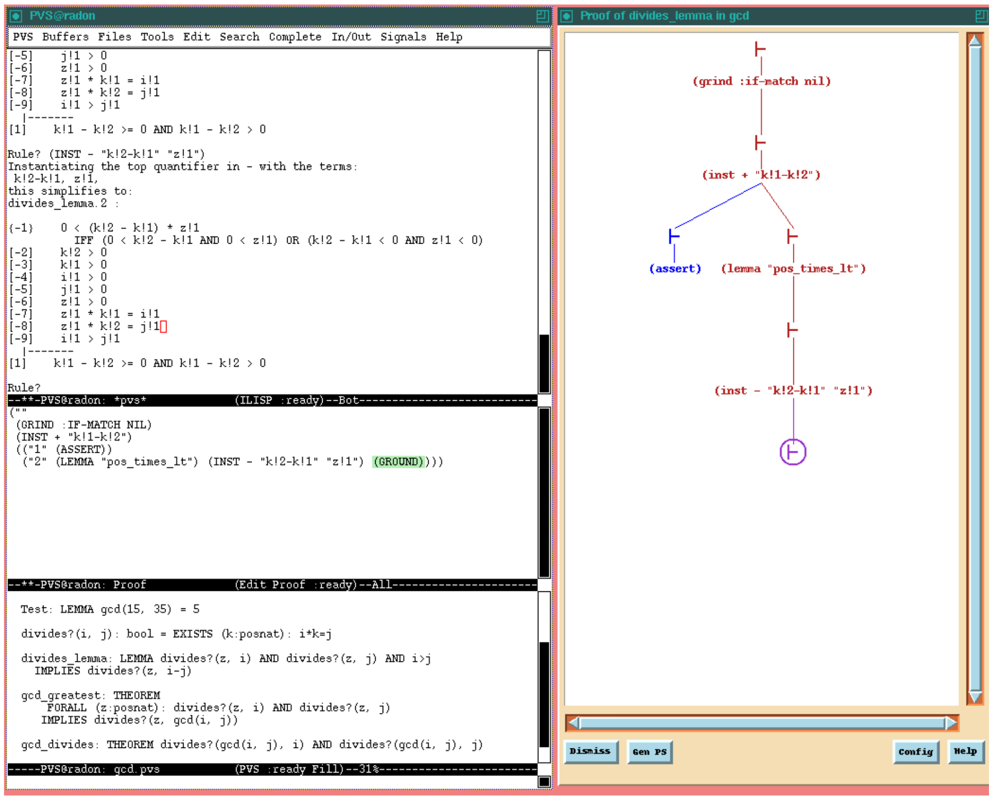
Prototype Verification System (PVS), which is available from SRI International, provides mechanized support for formal specification and verification. PVS is mainly intended for the formalization of requirements and design-level specifications, and for the analysis of intricate and difficult problems. It has been chiefly applied to algorithms and architectures for fault-tolerant flight control systems, and to problems in hardware and real-time system design.

PVS consists of a specification language, a number of predefined theories, a theorem prover, various utilities, documentation, and several examples that illustrate different methods of using the system in several application areas. PVS exploits the synergy between a highly expressive specification language and powerful automated deduction. For example, some elements of the specification language are made possible because the typechecker can use theorem proving. This distinguishing feature of PVS has allowed efficient treatment of many examples that are considered difficult for other verification systems.

The specification language of PVS is based on classical, higher-order logic. The base types include uninterpreted types that may be introduced by the user, and built-in types such as the Booleans, integers, reals, and the ordinals up to epsilon_0; the type-constructors include functions, sets, tuples, records, enumerations, and recursively-defined abstract data types, such as lists and binary trees. Predicate subtypes and dependent types can be used to introduce constraints, such as the type of prime numbers. These constrained types may incur proof obligations during type checking, but greatly increase the expressiveness and naturalness of specifications. In practice, the theorem prover discharges most of the obligations automatically. PVS specifications are organized into parameterized theories that may contain assumptions, definitions, axioms, and theorems. Definitions are guaranteed to provide conservative extension; to ensure this, recursive function definitions generate proof obligations. Inductively-defined relations are also supported. PVS expressions provide the usual arithmetic and logical operators, function application, lambda abstraction, and quantifiers, within a natural syntax. Names may be freely overloaded, including those of the built-in operators such as AND and +. Tabular specifications of the kind advocated by Parnas are supported, with automated checks for disjointness and coverage of conditions. An extensive prelude of built-in theories provides hundreds of useful definitions and lemmas; user-contributed libraries provide many more.

The PVS theorem prover provides a collection of powerful primitive inference procedures that are applied interactively under user guidance within a calculus framework. The primitive inferences include propositional and quantifier rules, induction, rewriting, and decision procedures for linear arithmetic. The implementations of these primitive inferences are optimized for large proofs: for example, propositional simplification uses BDDs, and auto-rewrites are cached for efficiency. User-defined procedures can combine these primitive inferences to yield higher-level proof strategies. Proofs yield scripts that can be edited, attached to additional formulas, and rerun. This allows many similar theorems to be proven efficiently, permits proofs to be adjusted economically to follow changes in requirements or design, and encourages the development of readable proofs. PVS includes a BDD-based decision procedure for the relational mu-calculus and thereby provides an experimental integration between theorem proving and CTL model checking.

PVS 3.0 is currently available only for Sparc machines with Solaris 2 and Intel x86 Machines with Linux compatible with Redhat 5 or later. The system is freely available under license from SRI International. PVS uses Gnu or X Emacs to provide an integrated interface to its specification language and prover. Commands can be selected either by pull-down menus or by extended Emacs commands. Extensive help, status-reporting and browsing tools are available, as well as the ability to generate typeset specifications (in user-defined notation) using LaTeX. Proof trees and theory hierarchies can be displayed graphically using Tcl/Tk (Figure 3-28). PVS is a large and complex system and it takes a long time to learn to use it effectively. One should be prepared to invest six months to become a moderately skilled user (less if one already knows other verification systems; more, if one needs to learn logic or unlearn Z). There are manuals, tutorials, and help available on the SRI International PVS website, http://pvs.csl.sri.com/.



```
[-5]    j!1 > 0
[-6]    z!1 > 0
[-7]    z!1 * k!1 = i!1
[-8]    z!1 * k!2 = j!1
[-9]    i!1 > j!1
  |-------
[1]    k!1 - k!2 >= 0 AND k!1 - k!2 > 0

Rule? (INST - "k!2-k!1" "z!1")
Instantiating the top quantifier in - with the terms:
 k!2-k!1, z!1,
this simplifies to:
divides_lemma.2 :

{-1}    0 < (k!2 - k!1) * z!1
          IFF (0 < k!2 - k!1 AND 0 < z!1) OR (k!2 - k!1 < 0 AND z!1 < 0)
[-2]    k!2 > 0
[-3]    k!1 > 0
[-4]    i!1 > 0
[-5]    j!1 > 0
[-6]    z!1 > 0
[-7]    z!1 * k!1 = i!1
[-8]    z!1 * k!2 = j!1
[-9]    i!1 > j!1
  |-------
[1]    k!1 - k!2 >= 0 AND k!1 - k!2 > 0

Rule?
--**-PVS@radon: *pvs*          (ILISP :ready)--Bot---------------------------
(""
 (GRIND :IF-MATCH NIL)
 (INST + "k!1-k!2")
 (("1" (ASSERT))
  ("2" (LEMMA "pos_times_lt") (INST - "k!2-k!1" "z!1") (GROUND))))
--**-PVS@radon: Proof          (Edit Proof :ready)--All---------------------

  Test: LEMMA gcd(15, 35) = 5

  divides?(i, j): bool = EXISTS (k:posnat): i*k=j

  divides_lemma: LEMMA divides?(z, i) AND divides?(z, j) AND i>j
    IMPLIES divides?(z, i-j)

  gcd_greatest: THEOREM
      FORALL (z:posnat): divides?(z, i) AND divides?(z, j)
     IMPLIES divides?(z, gcd(i, j))

  gcd_divides: THEOREM divides?(gcd(i, j), i) AND divides?(gcd(i, j), j)
-----PVS@radon: gcd.pvs          (PVS :ready Fill)--31%----------------------
```

Figure 3-28. PVS Screen Shot

Collaborative projects involving PVS are ongoing with NASA and several aerospace companies; applications include a microprocessor for aircraft flight-control, diagnosis and scheduling algorithms for fault-tolerant architectures, and requirements specification for portions of the Space Shuttle flight-control system. PVS has been installed at hundreds of sites in North America, Europe, and Asia. Currently, work is being done to develop PVS methodologies for highly automated hardware verification (including integration with model checkers), and for concurrent and real-time systems (including a transparent embedding of the duration calculus).

3.3.12 Real-Time Software Testing Tool Suite

Also known as Dr. Yann-Hang Lee's tool, the Real-Time Software Testing Tool Suite was developed for NASA IV&V under the OSMA program. It is a two-stage approach to minimize temporal interference during testing analysis [Lee 2000]. The first stage records interactions with the embedded environment and in the second stage replay allows deterministic execution combined with program instrumentation. The tool employs a deterministic replay mechanism that can insert unlimited volume of program instrumentation codes in tested real time software while guaranteeing the same behavior. It does this by logging program events that can be re-applied during program execution to replicate prior program behavior.

The software instruction counter has a weakness of intolerable probe effect overhead for real-time systems, from 10% to 20%. The author has developed an enhanced software instruction counter (ESIC) that reduces the probe effect overhead. This enhanced software instruction counter is coupled with *X*Suds software testing tool suite (Telecordia) used for cover analysis. While this run-time monitoring tool is not currently being used to test NNs, the developer believes that it may be able adapted for such a purpose.

The target platforms include Wind River's VxWorks operating system and the PowerPC architecture. The host platform uses Microsoft Visual C++, *X*Suds, and MS Windows NT. The implementation can be applied easily to other platforms.

The instruction counter that tracks software operation is one method that can reproduce exact behaviors when replaying the real-time applications. The new method Dr. Lee proposes reduces the probe effect by distinguishing between non-deterministic scope (NDS) and deterministic scope (DS) regions of the program by analyzing the source code. This allows for the removal of the record and replay instrumentation codes from the DS regions to reduce the probe effect overhead. All high-level language constructs such as while, for, if-then-else, and assignment will be tagged as either a non-deterministic language construct (NDLC) or a deterministic language construct (DLC). After classifying all language constructs in the program, *scope analysis* is done and the program is partitioned into non-deterministic scope (NDS) and deterministic scope (DS). This is illustrated in Figure 3-29.

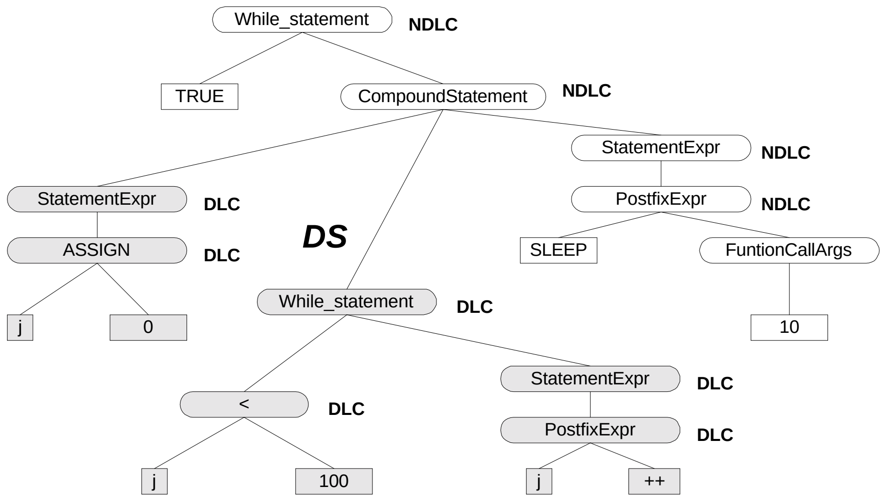


Figure 3-29. Language Construct Classified and Scope Analyzed Abstract Syntax Tree

A proof of the correctness of the ESIC method shows that the behavior of real-time software is not changed in replay mode execution.

The tool suite is composed of four functional units:

1. The Program Analyzer distinguishes between DS and NDS regions of the program. It is composed of a
   - Parser for syntax and semantic analysis,
   - Language construct classifier that classifies every statement as either DS or NDS,
   - Scope analyzer that partitions the code into DS and NDS and inserts scope marks for the ESIC record and replay, and a
   - Code generator that outputs the program source including scope marks.
2. The Enhanced Software Instruction Counter (ESIC) Record- and Replay-Instrumenter parses the assembly output from the program analyzer and puts record instrumentation codes before the three types of branch instructions. Replay instrumentation inserts replay codes into the program in the NDS regions. ESIC has been developed for the PowerPC platform. It is a second-pass PowerPC assembly translator. It reads the assembly file line by line and makes a symbol table that stores labels and their corresponding location. The assembly operation of backward branch is ESIC instrumented. Then it reads the assembly file again and searches for backward branch, jump, and subroutine calls. As illustrated in Table 3-10, the ESIC inserts record or replay codes before and/or after the found instructions.

Table 3-10. Record- and Replay- Instrumentation Examples

Instruction without condition codes	Record- and replay- instrumented code

label:
    …
    bla   sub1
    …
	label:
	    …
	    subi r14,r14,1
	    cmpwi r14,0
	    bne .LESIC_001
	    bla esic_handler
	.**LESIC_001:**
	    bla sub1
	    …

Instruction with condition codes	**Record- and replay- instrumented code**

label:
    …
    bge   label
    …
	label:
	    …
	    blt .LESIC_001
	    subi r14,r14,1
	    cmpwi r14,0
	    bne .LESIC_002
	    bla esic_handler
	.**LESIC_002:**
	    b label
	.LESIC_001:
	    …

3. The Event Logger and Replayer History Log File Converter is composed of Software Instruction Counter (SIC), Program Counter (PC), and event-specific information. It matches program counters between record- and program- & replay- instrumented codes using all available information such as linkage map file, linking format, etc.
4. The Software Testing Program Instrumentation uses the events logged in execution to replay the application deterministically. With the exact mapping of the logged temporal and spatial location of the events, post-software analysis is achieved.

The tool is being developed to test real-time systems and attempts to limit overhead by using the ESIC method for record and replay. The tool is still in development stages and is not available for distribution at this time.

3.3.13 RULEX

RULEX is a tool which can extract symbolic “if...then” rules by analyzing the underlying structure of a specific kind of back-propagation NN. The goal is to use these symbolic rules to provide insight into the decision making process of the network.

RULEX is more suited to fixed NNs. It might also be useful for networks undergoing training that can be held static between training sessions. This could be done to ensure the network is tending towards a correct operation. RULEX would not be very useful for quickly adapting systems.

RULEX works as a decompositional rule extraction tool, analyzing the network neuron-by-neuron to construct the symbolic rules. These symbolic rules take the form of:

IF Condition 1 **AND** Condition 2 **AND** Condition 3 **AND** … **THEN TRUE**

The RULEX tool is designed for the Constrained Error Back-Propagation (CEBP) network, a specific implementation of a multiplayer perceptron.

The hidden layers of the CEBP each represent a disjoint segment of the input training space. The neurons, which make up a hidden layer, are sigmoid-based locally responsive units (LRUs).

Each unit in the hidden layer has a defined region of operation from within the input space. The LRUs are explained as being composed of a set of ridges, one ridge for each dimension of the input. The one-dimensional ridge for an LRU is seen in Figure 3-30. An example of an LRU activation region across two dimensions is seen Figure 3-31. CEBP networks are capable of handling multi-dimensional data with the LRU regions created through the superposition of a ridge for each dimension of the input.

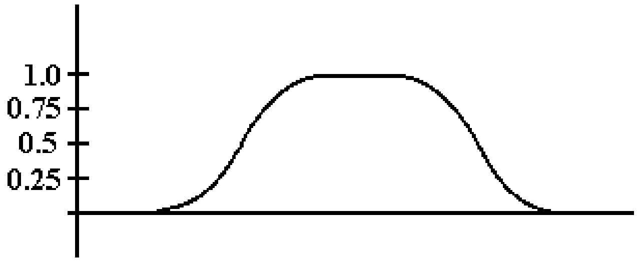


Figure 3-30. A 2D Ridge Representative of the Response Area of an LRU

The LRU will produce an output only if the value presented to the unit falls within the active range of the ridge.

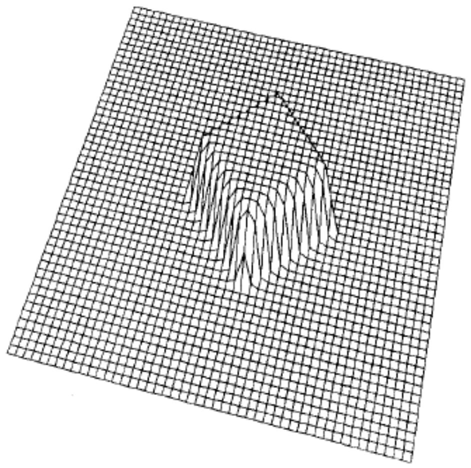


Figure 3-31. A 3D Ridge Representative of the Responsive Area of an LRU

For rule extraction to occur, each data point from within an input space must be classified by a single LRU. When the network is presented with an input stimulus, only the one LRU that corresponds to the input stimulus will generate an output.

The output of a multi-dimensional LRU is the thresholded sum of all of the outputs of its ridge components. The output function is such that each individual ridge must be 'active' for the LRU to be 'active.' Consequently, the LRU will not generate an output unless each dimensional component to the input stimulus falls within the ridges of the LRU.

Since all of the LRU ridges must be active to have the LRU active, the prepositional "if…then" rules can be extracted from the LRUs. As an example:

IF	RIDGE$_1$ is Active
AND	RIDGE$_2$ is Active
AND	RIDGE$_N$ is Active
THEN	Input Pattern is in the Target Class

The CEBP network can be constructed to work for discrete or continuous systems. The form of the rules stays the same but the conditional expressions change based upon the system type.

For a discrete system, the rules assume the form:

IF	V_{1a} **OR** V_{1b} … **OR** V_{1n}
AND	V_{2a} **OR** V_{2b} … **OR** V_{2n}
AND	V_{Na} **OR** V_{Nb} … **OR** V_{Nn}
THEN	*Input Pattern is in the Target Class*

where V_{ix} is the value along the i^{th} dimension and *a, b, .., n* are the possible states of that value.

For a continuous system, the rules assume the form:

IF	$c_1 - b_1 + 2.45k_1^{-1} \leq x_1 \leq c_1 + b_1 - 2.45k_1^{-1}$
AND	$c_2 - b_2 + 2.45k_2^{-1} \leq x_2 \leq c_2 + b_2 - 2.45k_2^{-1}$
AND	$c_N - b_N + 2.45k_N^{-1} \leq x_N \leq c_N + b_N - 2.45k_N^{-1}$
THEN	*Input Pattern is in the Target Class*

where c_i is the coordinate of the center of the ridge, b_i is the width of the ridge, and k_i is a steepness measure of the ridge - all variables used in the sigmoid equations that create the LRUs.

The relationship between the discrete and the continuous rules is $v_{ia} \geq c_i - b_i + 2.45k_i^{-1}$ and $v_{in} \leq c_i + b_i - 2.45k_i^{-1}$.

The RULEX tool is provided in the C language source code and an already compiled binary for use under MS-DOS on X86 platforms.

A description file is required for the network that provides the link between the numeric weights of the network and the symbolic output of the extracted rule. This is something that must be generated by the system analyst.

When run, the RULEX program parses the description file, a weight file for the network, and a test file name (containing test data to apply to the NN). The outputs from the execution are written to a file called "RULEX.out."

Since RULEX is a well-published algorithm and source code is provided for an implementation under MS-DOS, it is a free program.

A system developer with a good knowledge on the construction of the CEBP (or similar network) would not have a difficult time developing the description file for use with RULEX. Creation of the description file can require a large amount of effort.

When all of the files are in place, it is quite easy to run the program and inspect results.

Currently, RULEX only works with networks of the CEBP architecture. There are other networks that exhibit the same "one neuron activates at a time" form of operation, such as some RBF networks and SOMs. The RULEX method can be modified to allow these networks to benefit from this kind of rule extraction.

After the program has been instantiated, the rule generation routine runs without need of user input. All rules generated by RULEX are stored in a single output file.

Neither the successes nor failures of RULEX are well documented; only a few examples are provided in the literature. These examples include the successful application to a MONK's Robot Classification problem.

Finding the source code for RULEX required a great deal of effort, as the tool is not being updated. However, a small readme.txt file is included with the source code and gives an adequate, if brief, overview on using the tool.

3.3.14 SMV/NuSMV

Symbolic Model Verifier (SMV) is a Binary Decision Diagram (BDD)-based model checker developed by Ken McMillan at Carnegie Mellon University (CMU). Its purpose is to check finite state systems against specifications in Computation Tree Logic (CTL) [McMillian 2001]

More than a theoretical tool, SMV identified several minor bugs in the Livingstone health monitoring system employed on Deep Space One's Remote Agent, an autonomous spacecraft controller developed jointly by NASA Ames Research Center and the Jet Propulsion Laboratory [Pecheur 2000]

An input file is required that defines the model (Kripke structure) to be verified and its specifications. The model definition is written in an input language that McMillan designed to allow for the description of both synchronous system models and asynchronous networks of abstract, non-deterministic processes [McMillian 2001]. Using this language, a model can be constructed in modular, reusable components that can have a hierarchical organization. Booleans, scalars, and fixed arrays are the only data types used in SMV since it defines finite state systems. The model specification is written as a CTL formula in the input file (defined by the `SPEC` keyword). SMV will verify that all possible initial states of the model satisfy the specification.

The SMV package containing necessary software and supporting documentation is free and downloadable from the CMU School of Computer Science website: http://www-2.cs.cmu.edu/~modelcheck/smv.html.

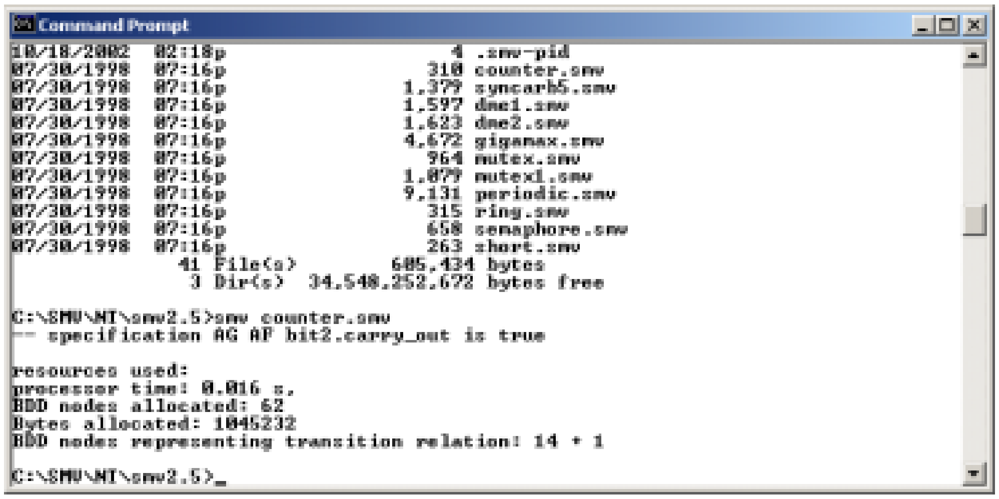
SMV only runs in batch mode on Windows NT and most UNIX based operating systems. It does not have a graphical user interface. It uses the DOS command prompt when run under Windows NT and the familiar textual interaction shell in UNIX. Installation under either requires little more than unpacking a compressed file to a directory and setting up the system path. Once installed, SMV can be used to verify included example models and CTL specifications.

Below is an example given by McMillian to show how SMV is used. Figure 3-32 is an SMV input file that models a 3-bit binary counter circuit. Note the CTL formula after the `SPEC` keyword. Figure 3-33 is a screen snapshot of the SMV output.

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
  AG AF bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := value + carry_in mod 2;
DEFINE
  carry_out := value & carry_in;
```

Figure 3-32. SMV Input File That Models 3-Bit Binary Counter Circuit



```
10/18/2002  02:18p                  4 .smv-pid
07/30/1998  07:16p                310 counter.smv
07/30/1998  07:16p              1,379 syncarb5.smv
07/30/1998  07:16p              1,597 dme1.smv
07/30/1998  07:16p              1,623 dme2.smv
07/30/1998  07:16p              4,672 gigamax.smv
07/30/1998  07:16p                964 mutex.smv
07/30/1998  07:16p              1,079 mutex1.smv
07/30/1998  07:16p              9,131 periodic.smv
07/30/1998  07:16p                315 ring.smv
07/30/1998  07:16p                658 semaphore.smv
07/30/1998  07:16p                263 short.smv
              41 File(s)        605,434 bytes
               3 Dir(s)  34,540,252,672 bytes free

C:\SMV\NT\smv2.5>smv counter.smv
-- specification AG AF bit2.carry_out is true

resources used:
processor time: 0.016 s.
BDD nodes allocated: 62
Bytes allocated: 1045232
BDD nodes representing transition relation: 14 + 1

C:\SMV\NT\smv2.5>_
```

Figure 3-33. SMV Output Screenshot After Executing Sample Model

It is unclear how much end-user support is provided for this tool, however bug fixes are provided on the CMU website. The site provides neither a frequently-asked-questions page nor a method to obtain assistance. Though the email addresses of the model checking group members are provided, it does not explicitly or implicitly say that those are for support purposes.

Though SMV was designed for non-deterministic and other types of systems, it was not designed for adaptive systems. Therefore, applicability to NNs would be limited to PTNNs.

NuSMV

NuSMV is an extension of SMV developed jointly by the Center for Technological and Scientific Research (ITC-IRST), CMU, University of Genova, and University of Trento. In addition to the capabilities inherited from SMV, NuSMV extends SMV by integrating model-checking techniques based on prepositional satisfiability (SAT). [Cimatti 2002] The complimentary model checking techniques of SAT and BDD, which solve different classes of problems, are combined and used in NuSMV.

Similar to SMV, NuSMV requires an input file that defines the model to be verified and its specifications. The language used to write this file is an extension of the SMV language. Unlike SMV, NuSMV runs in either batch mode or interactively via a text shell.

The main features of NuSMV as described in the NuSMV 2.1 User Manual are the following:

- Functionalities. SMV is expanded to allow for the analysis of specification formulas written in both CTL and LTL.
- Architecture. Effort needed to extend or modify NuSMV is eliminated by the defined software architecture.
- Quality of the implementation. It "is written in ANSI C, is POSIX compliant, and has been debugged with Purify in order to detect memory leaks" [NuSMV 2002].

Because NuSMV is developed and distributed with an open source license, anyone can use the tool and participate in its future development.

Support for NuSMV is much better defined than that of SMV, with more extensive documentation, tutorials, a bug submission website, and email contacts.

Finally, NuSMV is no more applicable to adaptive systems than is its predecessor SMV. Like SMV, its applicability to NNs is limited to PTNNs.

3.3.15 SPIN

SPIN (for Simple PROMELA INterpreter) is a scalable, finite state model checker application useful for verifying a multi-threaded plan execution programming language. It is a one-pass, on-the-fly verification tool that has its roots in a program developed at Bell Labs in 1980. The tool checks for the logical consistency of a specification, deadlocks, unspecified receptions, flag incompleteness, race conditions, and unwarranted assumptions about the relative speeds of processes [SPIN 2002].

SPIN uses a PROMELA, a high level language, to specify system descriptions. It works on-the-fly, so no construction of a global state graph or Kripke structure is required. It can be used as a full LTL model checking system, although basic safety and liveness properties do not require LTL. It supports rendezvous and buffered message passing, and communicates through shared memory.

SPIN offers three modes of operation:

- Simulation, for rapid prototyping with random, guided, or interactive simulations

- Exhaustive state space analyzer, proving user-specified correctness requirements using partial order reduction theory to optimize the search
- Bit-state space analyzer

Some of its noted success factors are

- Press-the-button verification
- Efficient implementation
- Good GUI (Xspin)
- More than two decades of research on advanced computer-aided verification, many relating to optimization algorithms [Ruys 2002].

SPIN is written in ANSI C. It can run on all UNIX versions and can also be compiled to run on Linux, Windows 95/98, and WindowsNT.

SPIN is well supported. In addition to online manual pages and documentation distributed with the application, support also exists from a variety of other sources. One of the richest sites is www.spinroot.com, which includes links to a tutorial, papers, and workshops.

3.3.16 STeP

Stanford Temporal Prover (STeP) is being developed by Stanford University's REACT research group to support the computer-aided formal verification of reactive, real-time and hybrid systems based on their temporal specification [Bjørner 1996].

STeP combines deductive methods with algorithmic techniques to verify linear-time temporal logic specifications of reactive and real-time systems. It uses verification rules, verification diagrams [Manna 1994], automatically generated invariants, model checking, and a collection of decision procedures to verify finite- and infinite-state systems.

Unlike most systems for temporal verification, STeP is not restricted to finite-state systems, but combines model checking with deductive methods to allow the verification of a broad class of complex systems, including parameterized circuit designs, parameterized programs, and programs with infinite data domains.

The deductive methods of STeP verify temporal properties of systems by means of verification rules and verification diagrams. Verification rules are used to reduce temporal properties of systems to first-order verification conditions. In support of this process, STeP has implemented verification diagrams to provide a visual language for guiding, organizing, and displaying proofs. These diagrams enable the user to construct proofs hierarchically, starting from a high-level and proceeding incrementally, as necessary, through layers of greater detail.

Deductive verification almost always relies on determining, for a given program and specification, suitably strong auxiliary invariants and intermediate assertions. STeP implements a variety of techniques for automatic invariant generation. These methods include local, linear, and polyhedral invariant generation, which perform an approximate, abstract propagation through the system. Verification conditions can then be established using the automatically generated auxiliary invariants as background properties.

Finally, STEP also provides an integrated suite of simplification and decision procedures for automatically checking the validity of a large class of first-order and temporal formulas. This degree of automated deduction is intended to efficiently handle most verification conditions that arise in deductive verification. An interactive Gentzen-style theorem prover and a resolution-based prover are available to establish the verification conditions that are not proven automatically.

An overview of the STEP architecture is presented in Figure 3-34. The basic input is a reactive real-time system, which may include both hardware and software descriptions. The system is expressed as a fair transition system [Manna 1991]. System properties to be proven are represented by the temporal logic formula. User guidance can be provided as intermediate assertions or visually via the verification diagrams. In either case, the system is responsible for generating and proving all of the required verification conditions.

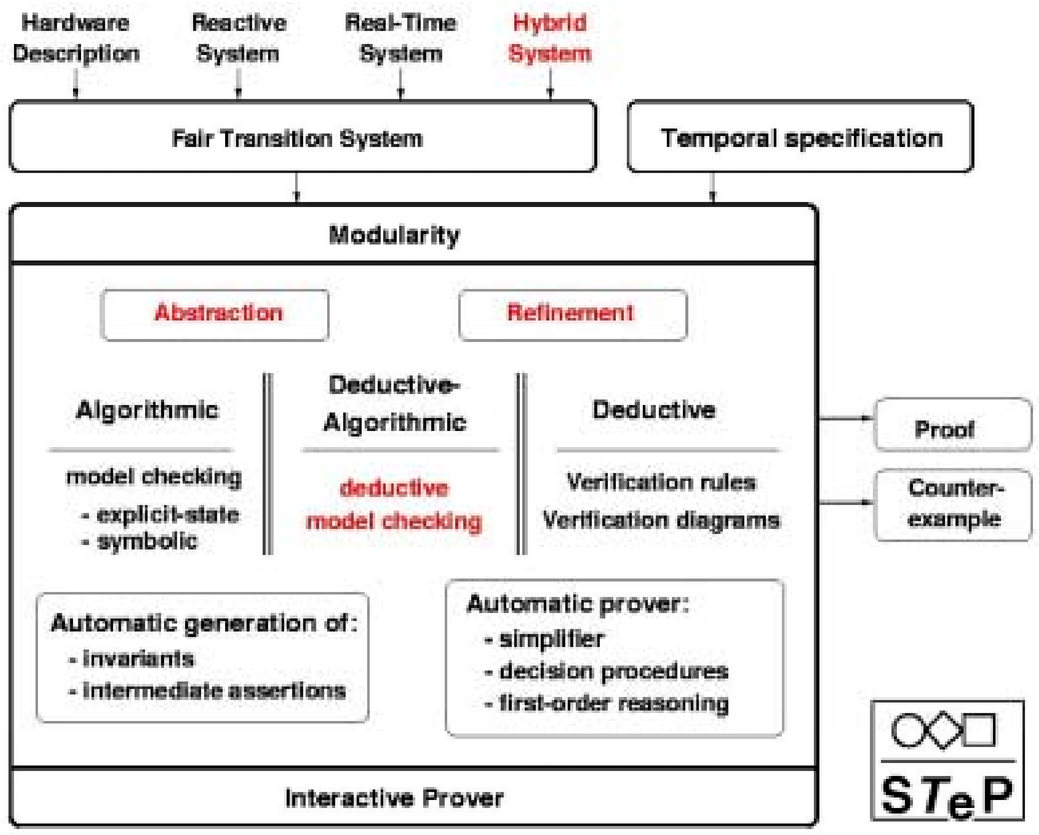


Figure 3-34. The STEP System Structure

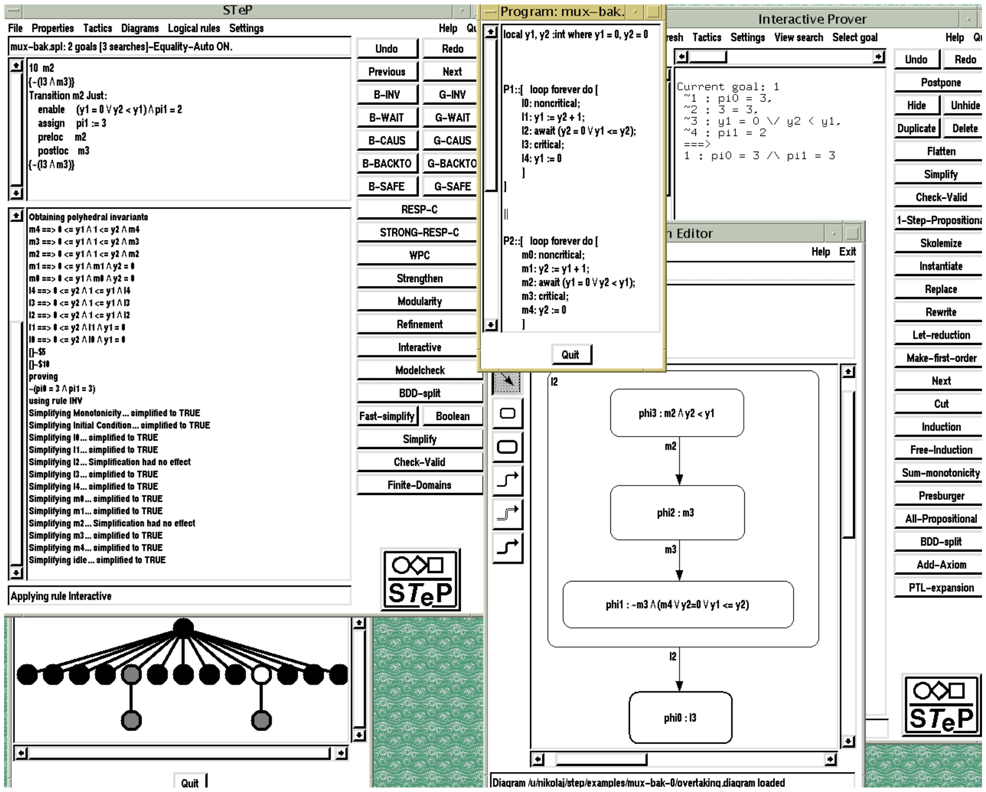
An educational version of the system, which accompanies the textbook [Manna 1991] is available. The distribution includes a comprehensive user manual [Manna 1995] and a tutorial. For many programs, ready-to-load verification diagrams are included as well.

STEP has three main interface components:

1. Top-level Prover, from which verification sessions are managed and verification rules are invoked
2. Interactive Prover, used to prove the validity of first-order and temporal-logic formulas that are not proven automatically

3. Verification Diagram Editor*,* for the creation of Verification Diagrams.

Figure 3-35 shows these three interfaces, with a version of the Bakery algorithm loaded, together with a tree representing the ongoing proof process.



```
mux-bak.spl: 2 goals [3 searches]-Equality-Auto ON.

10  m2
{~(l3 ∧ m3)}
Transition m2 Just:
   enable    (y1 = 0 ∨ y2 < y1) ∧ pi1 = 2
   assign    pi1 := 3
   preloc    m2
   postloc   m3
{~(l3 ∧ m3)}
```

```
Obtaining polyhedral invariants
m4 ==> 0 <= y1 ∧ 1 <= y2 ∧ m4
m3 ==> 0 <= y1 ∧ 1 <= y2 ∧ m3
m2 ==> 0 <= y1 ∧ 1 <= y2 ∧ m2
m1 ==> 0 <= y1 ∧ m1 ∧ y2 = 0
m0 ==> 0 <= y1 ∧ m0 ∧ y2 = 0
l4 ==> 0 <= y2 ∧ 1 <= y1 ∧ l4
l3 ==> 0 <= y2 ∧ 1 <= y1 ∧ l3
l2 ==> 0 <= y2 ∧ 1 <= y1 ∧ l2
l1 ==> 0 <= y2 ∧ l1 ∧ y1 = 0
l0 ==> 0 <= y2 ∧ l0 ∧ y1 = 0
[]-$5
[]-$10
proving
~(pi0 = 3 ∧ pi1 = 3)
using rule INV
Simplifying Monotonicity... simplified to TRUE
Simplifying Initial Condition... simplified to TRUE
Simplifying l0... simplified to TRUE
Simplifying l1... simplified to TRUE
Simplifying l2... Simplification had no effect
Simplifying l3... simplified to TRUE
Simplifying l4... simplified to TRUE
Simplifying m0... simplified to TRUE
Simplifying m1... simplified to TRUE
Simplifying m2... Simplification had no effect
Simplifying m3... simplified to TRUE
Simplifying m4... simplified to TRUE
Simplifying idle... simplified to TRUE

Applying rule Interactive
```

```
local y1, y2 :int where y1 = 0, y2 = 0

P1::[  loop forever do [
      l0: noncritical;
      l1: y1 := y2 + 1;
      l2: await (y2 = 0 ∨ y1 <= y2);
      l3: critical;
      l4: y1 := 0
      ]
]

||

P2::[  loop forever do [
      m0: noncritical;
      m1: y2 := y1 + 1;
      m2: await (y1 = 0 ∨ y2 < y1);
      m3: critical;
      m4: y2 := 0
      ]
```

```
Current goal: 1
 ~1 : pi0 = 3,
 ~2 : 3 = 3,
 ~3 : y1 = 0 \/ y2 < y1,
 ~4 : pi1 = 2
 ===>
 1 : pi0 = 3 /\ pi1 = 3
```

```
Diagram /u/nikolaj/step/examples/mux-bak-0/overtaking.diagram loaded
```

Figure 3-35. The STEP User Interface

STEP has been used to analyze a diverse number of systems, including:

- An infinite-state demarcation protocol used in distributed databases
- A pipelined four-stage multiplication circuit
- Ricart and Agrawala's mutual exclusion protocol
- Several (*N*-component) ring arbiters
- Szymanski's *N*-process mutual-exclusion algorithm
- An industrial split-transaction bus protocol to coordinate access for six processors.

Real-time systems analyzed include Fisher's mutual-exclusion protocol and a (parameterized) railroad gate controller [Hietmeyer 1994].

STEP is implemented in the programming language Standard ML of New Jersey, using CML and eXene for its X-Windows user interface.

3.3.17 UPPAAL

UPPAAL is a tool suite for verification of real-time systems that continues to be developed collaboratively by the Basic Research in Computer Science at Aalborg University in Denmark and the Department of Computer Systems at Uppsala University in Sweden. It is an integrated environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.).

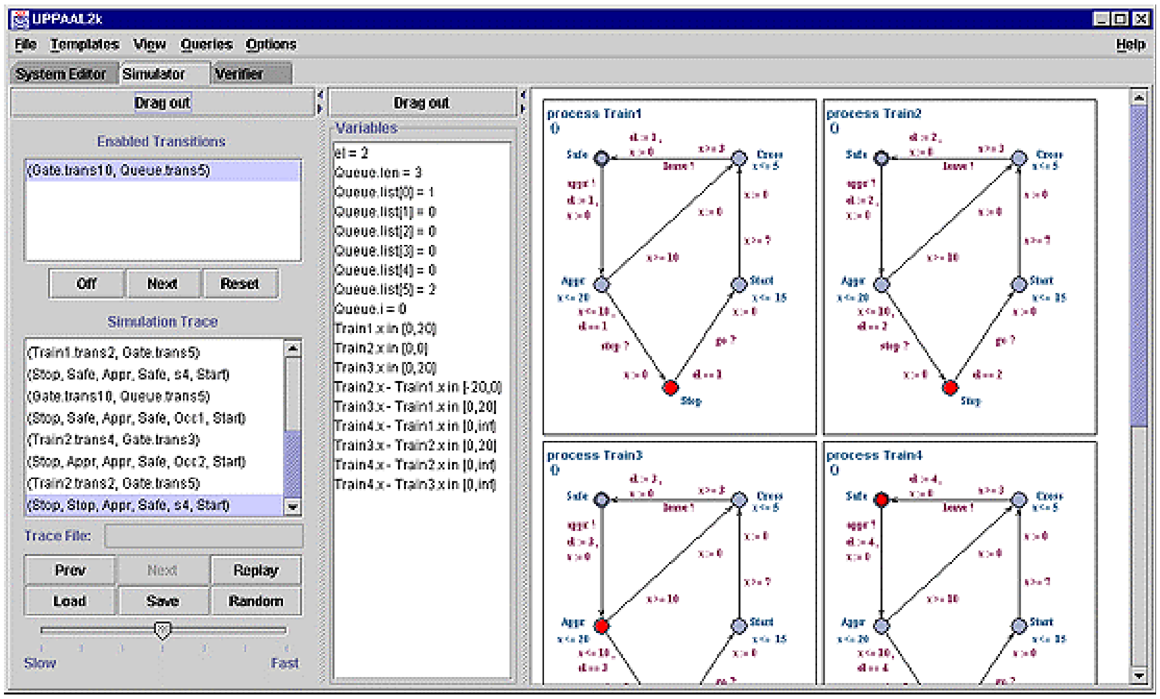
UPPALL is generally appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables. Typical application areas for UPPAAL include real-time controllers and communication protocols, in particular, those where timing aspects are critical.

UPPAAL consists of three main parts:

- The description language is a non-deterministic guarded command language with simple data types (e.g. bounded integers, arrays, etc.). It serves as a modeling or design language to describe system behavior as networks of automata extended with clock and data variables.
- The simulator validation tool enables examination of possible dynamic executions of a system during early design stages. This provides an inexpensive means of fault detection prior to verification by the model-checker.
- The model-checker checks invariant and reachability properties by exploring the state-space of a system, i.e. reachability analysis in terms of symbolic states represented by constraints.

The simulator and the model-checker are designed for interactive and automated analysis of system behavior by manipulating and solving constraints that represent the state-space of a system description. They share a common basis: constraint-solvers.

UPPALL provides both graphical and textual formats for the description language – one for easy user interactive use and the other for ready interaction with automation of preprocessors, etc. The description language, in particular, supports hybrid automata where the behavior of the system variables can be described or approximated using lower and upper bounds on their rates. A screen snapshot of UPPALL2k, the most recent version of UPPALL, is shown in Figure 3-36.



```
(Gate.trans10, Queue.trans5)
```

```
(Train1.trans2, Gate.trans5)
(Stop, Safe, Appr, Safe, s4, Start)
(Gate.trans10, Queue.trans5)
(Stop, Safe, Appr, Safe, Occ1, Start)
(Train2.trans4, Gate.trans3)
(Stop, Appr, Appr, Safe, Occ2, Start)
(Train2.trans2, Gate.trans5)
(Stop, Stop, Appr, Safe, s4, Start)
```

```
el = 2
Queue.len = 3
Queue.list[0] = 1
Queue.list[1] = 0
Queue.list[2] = 0
Queue.list[3] = 0
Queue.list[4] = 0
Queue.list[5] = 2
Queue.i = 0
Train1.x in [0,20]
Train2.x in [0,0]
Train3.x in [0,20]
Train2.x - Train1.x in [-20,0]
Train3.x - Train1.x in [0,20]
Train4.x - Train1.x in [0,inf]
Train3.x - Train2.x in [0,20]
Train4.x - Train2.x in [0,inf]
Train4.x - Train3.x in [0,inf]
```

Figure 3-36. UPPAAL2k on Screen.

The features of UPPAAL2k include:

- A graphical system editor that allows graphical descriptions of systems.
- A graphical simulator, which provides graphical visualization and the ability to record the possible dynamic behaviors of a system description. It may also be used to visualize traces generated by the model-checker.
- A requirement specification editor that also constitutes a graphical user interface to the verifier of UPPAAL2k.
- A model-checker for automatic verification of safety and bounded-liveness properties by reachability analysis of the symbolic state-space.
- Generation of diagnostic traces in case verification of a particular real-time system failure. The diagnostic traces may be automatically loaded and graphically visualized using the simulator.

The two main design criteria for UPPAAL have been efficiency and ease of use. The application of an on-the-fly searching technique has been crucial to the efficiency of the UPPAAL model-checker. Another important key to efficiency is the application of a symbolic technique that reduces verification problems to that of efficient manipulation and solving of constraints.

UPPAAL has been applied successfully in a number of industrial case studies, for example:

- **Bounded Retransmission Protocol:** This protocol is based on the alternating bit protocol over a lossy communication channel, but allows for a bounded number of retransmissions. D'Argenio reported that a number of properties of the protocol is automatically checked with UPPAAL [D'Argenio 1997]. In particular, it is shown that the correctness of the protocol is dependent on correctly chosen time-out values.
- **Collision Avoidance Protocol:** The protocol in [Jensen 1996; Aceto 1998] is implemented on top of an Ethernet-like medium such as the CSMA/CD protocol. It is used to ensure an upper bound on the communication delay between the network nodes. It was designed and proven correct using UPPAAL. UPPAAL showed that the protocol is collision-free, and that it does ensure an upper bound on the user-to-user communication delay (assuming a perfect medium).
- **LEGO MINDSTORMS Systems - Verification of RCX Systems**: The studied problem is that of checking properties of actual programs, rather than abstract models of programs. It is shown how UPPAAL models can be automatically synthesized from RCX programs, written in the programming language Not Quite C (NQC). The system is modeled and checked using UPPAAL2k.
- **Multimedia Stream:** Bowman presents the specification and verification of a multimedia stream in UPPAAL [Bowman 1998]. Multimedia streams are the building blocks of distributed multimedia applications. The stream is described in the UPPAAL model and then certain real-time properties are verified in the model-checker. Verification of throughput and end-to-end latency are the primary focus.
- **Philips Audio Protocol with Bus Collision:** This is an extended variant of Philips audio control protocol with bus collision detection. Its correctness was originally proven by hand, and later proven by using UPPAAL in [Behrmann 1999].

UPPAAL is a client/server application implemented in Java and C++, and is currently available for Linux, SunOS and Windows 95/98/NT. Subject to some conditions, it is free for non-profit applications.

The developers of UPPAAL make no commitment to support the product and do not guarantee the results it produces. It is available as research tool for other researchers to build upon.

3.3.18 WVU F-15 Simulation

The WVU F-15 Simulation is a simulation tool developed for the IFC program by researchers in the Department of Mechanical and Aerospace Engineering at WVU. Dr. Marcello Napolitano has led development of this tool. This simulation has been described as a flexible environment for analyzing different research issues within the flight control system of the IFC program mentioned in Section 3.1.2.1 [Perhinschi 2002]. This tool allows for detailed analysis of the different research components that comprise the Intelligent Flight Control System, and, therefore, may serve as a useful tool for the study of the NN components within this project. The WVU F-15 Simulation can be considered a tool that addresses both testing and visualization of the software.

The simulation environment is MATLAB/Simulink. The simulation incorporates the NN components including the fixed PTNN and adaptive OLNN. While the simulation is specific for the F-15, the researchers at WVU have proven they can modify the internal data to accommodate additional aircraft such as the DeHavilland 2 Beaver, a propeller fixed-wing aircraft.

The visualization queues are provided through the Aviator Visual Design Simulator (AVDS) 3D visualization package that has been integrated into the MATLAB simulation. The AVDS package is a simple but effective 3D representation of the aircraft and offers different viewing points both external and internal to the vehicle. It can also display traditional pilot instrumentations, such as altitude and flight direction, for visual pilot feedback.

Real-time MATLAB plots are generated during the flight and are displayed on the screen or stored for later analysis. These plots are user selected and show various values including sensor data, error tracking of the research components, and pilot input. This is shown in Figure 3-37.

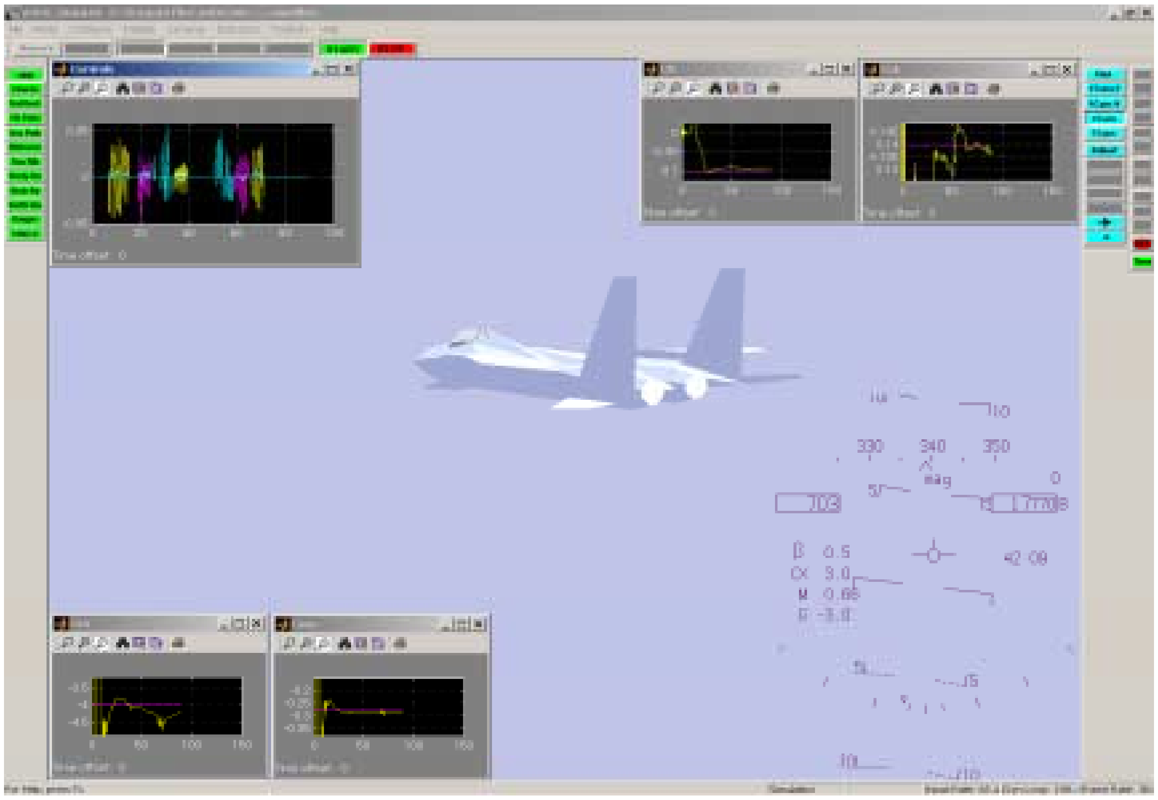


Figure 3-37. The WVU F-15 Simulation

The simulation package allows the user to choose between several options through menus in a graphical user interface (GUI) as seen in Figure 3-38. Perhinschi documents that the menus are simple, easy to use, and allow for the selection of several simulation conditions including:

- Nominal or failure conditions
- Characteristics of the induced failure
  - Time of occurrence
  - Surface affected
  - Locked surface, missing surface, combination of both
  - Position locked
  - Percentage missing surface
- Origin of input (joystick or pre-recorded commands)
- Research component versions (allowing different PTNNs, different OLNNs, etc.)
- Visualization and output options [Perhinschi 2002].

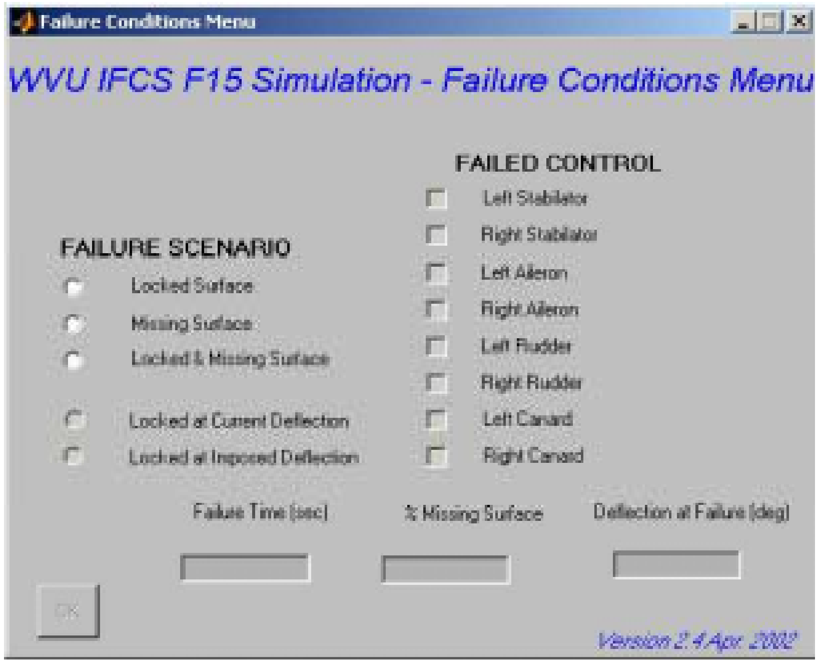


Figure 3-38. WVU Menus

The simulation was first developed for the GEN1 architecture, but further work has been done to create the early version of the GEN2 architecture. Within the different architectures, WVU has designed a process of selection that allows for choice of different NNs types. For example, the online learner can be a dynamic cell structure, a modified RBF network, or anything that a system user may wish to add into the simulation.

The flight simulation works within a Windows/PC hardware configuration. MATLAB is available across other platforms including Macintosh, UNIX, and Linux, but the AVDS 3D visualization is a Windows only based system. Continued development may allow for the integration of a 3D visualization package from MATLAB making the tool more platform independent.

The minimum hardware requirements for AVDS is a Pentium based PC running a version of Windows later than Windows 95 with 16 MB of RAM (32 MB for Windows NT) and at least 100 MB of disk space. However, more memory, more disk space, and a graphics accelerator capable of hardware acceleration of OpenGL are recommended.

A drawback to the simulation is the research-oriented development that keeps it in an unclassified environment. The development has been done using public domain data and publications that do not contain the exact detail of the F-15 aircraft. The F-15 aerodynamic and thrust data used was from freely distributed code supplied by NASA for the 1990 AIAA GNC Design Challenge. This data was for an F-15 with a single rudder tail whereas the F-15 ACTIVE has the standard two rudder configuration. A detailed analysis was done on the WVU model, with the help of NASA DFRC engineers, to confirm that the WVU Simulation was similar to the actual F-15 model [Perhinschi 2002]. The simulation developers are very knowledgeable about the F-15 system, in general, and other non-military airframes can also be incorporated into the simulation.

The WVU Simulation requires MATLAB, Simulink, and AVDS. A single MATLAB license for a PC is approximately $5,000 (with all necessary toolboxes), Simulink is approximately $3,000, and AVDS is $5,000. The WVU code itself is not yet available for public release, but it was developed under a grant for NASA DFRC; therefore, it is obtainable for NASA IV&V research.

The tool makes use of a simple to understand GUI that, through button selection, allows for configuration of the simulation. Each GUI gives a small explanation of what is to be selected and is easy to follow. There are several GUI windows that a user must work through, but the total time and effort during the selection process is minimal.

The simulation uses the MATLAB language, which is like the C programming language. The components of the Intelligent Flight Control System, such as the NNs, must be in Simulink, which is more of a graphical programming language. MATLAB does provide a convenient method to translate C programs into Simulink language blocks. If the code is in another language, then it must be converted into C (which can then be easily moved into Simulink), or re-programmed directly into Simulink.

The WVU Simulation is not automated and requires a user to operate the joystick to control the aircraft. It also requires a user to select the scenarios each time through the simulation, although this can be changed to run with pre-recorded commands from a data file.

Both the GEN1 and an early version of the GEN2 controllers have been tested and the simulation does appear to be a useful tool based upon feedback from members of NASA DFRC.

MATLAB, Simulink, and AVDS have good technical support. WVU is still in the developing stages of the GEN2 scheme and have been very willing to make modifications and improvements based on user feedback.

4.0 EVALUATION OF TOOLS

A summary of the tools discussed and evaluated in this document is presented in the table below. The table provides evaluation characteristics for each tool, organized in eight columns: developer, applicability to NNs, expense, ease of use, translation required, automation, track record, and available support. The goal of this table is to aid the decision-maker in choosing the best tool for an application. With the exception of LOTOS, evaluation criteria were restricted according to the following legend:

- NN Applicability: Yes, No, Partially
- Expense: Commercial, Shareware, or Freeware
- Ease of Use: Expert, Intermediate, or Beginner
- Translation: Required, or Not Required
- Automation: Yes, No, or Partial
- Track Record: Successful, Proven Unsuccessful, or Unproven
- Support: Commercial, Formal, or Informal

Section	Tool Name	Developer	NN Applicability	Expense	Ease of Use	Translation	Automation	Track Record	Support
3.3.1	**HyTech**	University of California, Berkeley	Partially	Freeware	Intermediate	Not Required	Partial	Successful	Informal
3.3.2	**Java PathExplorer**	NASA ARC	No	N/A	Intermediate	Not Required	Yes	Successful	Informal
3.3.3	**Java PathFinder 2**	NASA ARC	No	Freeware	Intermediate	Not Required	Yes	Successful	Informal
3.3.4	**KRONOS**	VERIMAG Corp.	Partially	Freeware	Intermediate	Not Required	Partial	Successful	Informal
3.3.5	**LOTOS**	Twente University	No	Freeware	Intermediate	Required	Yes	Successful	Informal
3.3.6	**MATLAB NN Toolbox**	MathWorks	Yes	Commercial	Expert	Not Required	Yes	Successful	Commercial
3.3.7	**Murphi**	Stanford University	Partially	Freeware	Intermediate	Required	Yes	Successful	Informal
3.3.8	**PARAGON**	University of Pennsylvania	No	Freeware	Expert	Required	Partial	Unproven	None
3.3.9	**PAX**	University of Kiel	No	Freeware	Intermediate	Not Required	Yes	Successful	Informal
3.3.10	**Planview/Comview**	Reid Simmons and Gregory Whelan	No	Commercial	Intermediate	Not Required	Yes	Successful	Formal
3.3.11	**PVS**	SRI International	No	Freeware	Expert	Required	Yes	Successful	Formal
3.3.12	**Real-Time Testing Suite**	Yann-Hang Lee	No	Freeware	Intermediate	Not Required	Yes	Unproven	Informal
3.3.13	**RULEX**	Robert Andrews and Shlomo Geva	Yes	Freeware	Intermediate	Not Required	Yes	Unproven	Informatl

Section	Tool Name	Developer	NN Applicability	Expense	Ease of Use	Translation	Automation	Track Record	Support
3.3.14	**SMV/NuSMV**	Carnegie Mellon University	Partially	Freeware	Expert	Not Required	Partial	Successful	Informal
3.3.15	**SPIN**	Bell Labs	No	Freeware	Intermediate	Required	Yes	Successful	Informal
3.3.16	**STeP**	Stanford University	Partially	Freeware	Intermediate	Required	Yes	Successful	Informal
3.3.17	**UPPAAL**	Uppsala University in Sweden and Aalborg University in Denmark	Partially	Freeware	Intermediate	Required	Yes	Successful	Informal
3.3.18	**WVU F-15 Simulator**	WVU	Yes	Freeware	Intermediate	Not Required	Yes	Unproven	Informal

5.0 CONCLUSION

5.1 Method Review

Based on the literature survey and other investigation, six methods seem most promising for the verification and validation of NNs. Those methods consist of traditional and automated testing techniques, run-time monitoring, Lyapunov stability analysis, rule extraction, cross validation, and visualization. Only one method, model checking, appears to have limited application to NNs.

Testing is by far the easiest in practice. Neural network developers already separate input data into training and testing data sets for NN development. Traditional testing, however, fails to assure the rigorous standards required for high reliability environments and safety-critical systems. Since most NNs are not used in high-reliability environments, a testing set for error calculation is all a developer needs to assess the system. But for those high assurance systems, such as aviation and robotic exploration, applying a simple testing set does not work. Automated testing, in combination with developing novel test generation algorithms, can overcome some of the limitations of traditional testing.

Run-time monitoring is the logical vehicle to offer real-time control and assessment of NNs. The system monitor can act as an early warning system when an NN begins to behave incorrectly. It may also be configured to remove the NN from control and replace it with a backup, perhaps one with a standard programming technique. The next step in the IVVNN will be the development of tools for run-time monitoring. These tools work for both fixed and dynamic NNs, but their biggest payoff will be for the dynamic adaptive systems.

Lyapunov stability appears to have some promising results and could work well as a type of safety monitor that continually analyzes a network to determine if it is tending towards stability or convergence.

Rule extraction can provide a tester with insight into what a fixed NN has learned and let him determine the acceptability of the network. Extraction does not work as well for dynamic systems because the rules need to be extracted after each iteration of learning and then judged for correctness. However, the extraction techniques are similar to rule initialization and rule insertion, which are more applicable to dynamic NNs. Through rule initialization, the network is given a starting point from which to adapt, which may offer some improved confidence in its behavior. Rule insertion can be performed periodically, while in operation or offline, to steer a dynamic network towards an area of knowledge.

Cross validation may work similar to diversification for fault-tolerant software. Instead of using a single NN, diverse NNs would use several systems of different architectures. Through configuration, these NNs can work in combination, providing checks and balances to system control. The output from individual networks can be routed through a voting mechanism that selects a course of action based on either a comparison or the weighted average of responses from the networks. This technique filters out individual failures in an NN because it can rely upon a set of NNs. System reliability can be expected to improve because the likelihood of all of the NNs failing might be proven to be very low.

Visualization, which is commonly used to analyze software systems, may be especially helpful for understanding the complex behavior of an NN in IV&V efforts. It can graphically portray the way the weights and internal connections of an NN change, aiding human interpretation and analysis. A plot of a changing error function gives almost immediate clues as to how well an NN is learning.

One common method that does not seem to be useful for NN analysis is model checking. A model checker searches all possible state transitions and execution paths to search for violations of requirements. While some networks could be thought of as having states, it would be unrealistic to attempt to model an NN within this framework. For an NN, its function approximation is not an execution path. Model checking would fail to identify the input sequence states that control how a network learns.

5.2 Tool Review

Of the tools the ISR reviewed, very few were found to be directly applicable to NNs. Basic tools such as the MATLAB NN Toolbox are of benefit to NN developers and testers, but the tools do not offer guidance as to how the tester should analyze the system or what the results actually mean.

The techniques applied to validate and verify autonomous systems such as RAX will not translate well to NNs. These methods check for thread behavior and inconsistencies with resource utilization and undefined state responses. Tools like Java Pathfinder and Java PathExplorer analyze lock and semaphore access to eliminate potential deadlocks and race conditions. These are types of faulty behavior that can occur in almost any type of multiprocessing software system. These tools fail to address the unique characteristics of NNs.

Additional tools may exist that offer greater benefit for NN analysis than the ones reviewed for this report. The Future Work section of this report addresses how the ISR will improve its tool knowledge through use of some of these applications, perhaps on a sample set of NN code.

5.3 Summation

Overall, the V&V field for NNs is starting to receive attention; new tools and techniques have emerged in the past ten years. Still, with all of this effort, no easy way has been developed for an IV&V practitioner to know what will, or will not, work for a particular system. No methodology exists to provide a practitioner with the ability to IV&V an NN. The ISR has surveyed the most prevalent V&V techniques for NNs and is exploring possibilities for combining their use into a methodology that will be useful for the IV&V of NNs.

6.0 FUTURE RESEARCH

The literature summary was the first task performed by the ISR for Development of Methodologies for IVVNN. Having surveyed the literature concerning current methods and tools available, the ISR will address two issues:

- No overall standard exists that addresses independent verification and validation techniques specifically for NNs.
- Current V&V techniques for NNs are still immature and are not sufficiently developed.

The ISR will approach these issues through two avenues. The ISR will perform research in quantifying human pilot factors that may be useful in developing a methodology for IVVNN. The ISR will also build upon existing IV&V processes such as formal methods and testing, to develop a unique process effective for NNs.

6.1 Research on Human Factor Analysis Based on Pilot Certification

The ISR has proposed that the criteria used to evaluate human pilots for the military may be helpful in evaluating NNs for use in aerospace applications. Given that artificial NNs are mathematical simulations of biological intelligence, there are parallels that can be identified between an ANN used in intelligent flight control and a human pilot.

The next phase of the IVVNN project will evaluate military standards for pilot certification and investigate the answers to the following questions:

- Can studying the process by which a pilot is certified give insight into the process for developing and validating NNs?
- Can techniques used for training humans be extended to tools and methods used for training NNs?
- Is there a mapping between run-time monitoring and pilot performance evaluation, between rule extraction and pilot communication, or between automated testing and hours of pilot training in a cockpit?

The result of this study may lead to formulating an underlying IV&V standard for NNs. Methods evaluated in this report may become components of this standard. Techniques based on these methods may be developed and used for evaluating NNs just as processes and steps have been developed to evaluate pilots.

6.2 Improvements on Existing Processes

In addition to the human factors research, the ISR will also explore ways to utilize, advance, or adapt existing IV&V methods to create a process that is effective for analyzing NNs. The ISR will examine formal methods, run-time monitoring, testing, and visualization.

6.2.1 Formal Methods

The term "Formal methods" refers to the use of techniques from formal logic and discrete mathematics in the specification, design and construction of computer systems and software. The purpose of applying formal methods is to make verification and validation of the software more objective by supplementing the traditional testing methods. The more rigorous the formal method the more effort and skill required to apply it and the more assurance the method will provide. The two formal methods that the ISR intends to examine in detail for application to NNs are rule extraction and model checking.

6.2.1.1 Rule Extraction

Rule extraction techniques may be appropriate for fixed NNs like a PTNN. If an NN has already been trained and tested to acceptable levels by its development team, an IV&V practitioner could then apply rule extraction to produce rules. These rules could then be compared against the original set of requirements and would provide information for review of the correctness of the function the network is approximating. For situations where NNs have inadequate requirements, rule extraction can be used to generate reverse requirements of the knowledge contained in the NN. At a minimum, extraction of these rules would provide some sense of confidence that the network will behave as it was intended.

For NNs that are dynamic like the OLNNs, similar techniques, such as rule initialization and rule insertion would be more appropriate. Improvements of a network's generalization might be made from specifically setting it at a desired starting point. Setting up a starting point through rule initialization could lead to a constrained learning regime. Rule insertion could be applied to continually steer a network's learning back to this desired operational regime. As a network adapts in a system, it may tend to "forget" previous knowledge, rule insertion could serve as a memory reminder to keep the network within an input space.

Rule extraction and rule insertion applied with NN diversification may prove to be useful. In systems that make use of several different NN architectures for reliability enhancement, fault recovery may be accomplished through extraction rather then insertion. Consider a scenario where three differently configured SOMs work together to form a system. If one of the networks fails, the system could be designed to reset that faulty network through rule extraction and rule insertion, reloading knowledge from the other two NNs into the failed network. This could restore the original N-system configuration rather than degrading to a (N-1)-system.

The ISR will use the MLP PTNN and the SOM OLNN from the IFC project to investigate how rule extraction techniques work on complex networks. From the literature survey it appears that the rule extraction techniques have only been applied to simple NNs, mainly chosen for their suitability to these techniques. Rule extraction, using the thresholding approach to generate decompositional rules, relies on networks where the internal neurons represent distinct regions. Obviously, not every project will make use of these kinds of networks. The IFC project's NNs will be an excellent testbed to explore the possibilities of rule extraction, rule initialization and rule insertion. Since requirements were generated for both of the networks used in the IFC project, rule extraction might be used in a requirements traceability study. Further, we can observe the performance of the OLNN given an initial set of knowledge based upon *a priori* system information to see if this initialization improves overall stability and correctness.

6.2.1.2 Model Checking

Model checking on an individual NN component appears to be ill defined and complicated, if even possible. The ISR will explore how model checking may be applied to a high level view of a system that contains an NN. For this work, the ISR will examine existing model checking techniques to determine if they can be applied to the IFC system. This exploration will determine whether model checking will be useful in the IV&V process for NNs.

6.2.2 Run-Time Monitoring Methods

Run-time or operational monitoring methods appear to be the next evolution of IV&V for OLNNs. Aimed at application for dynamic NNs, a run-time monitor would have to be developed as part of the system during the design phase of the project. The monitor, working like an oracle, would provide system specific solutions. The ISR will investigate at least four classes of run-time monitors.

6.2.2.1 Data Sniffing (WVU)

Data sniffing is used to assess data as it enters and exits an NN to determine whether the OLNN has adjusted acceptably. This is an area of ongoing research at WVU. The ISR will assess the benefits of data sniffing by applying this technique to the OLNN of the IFC project.

6.2.2.2 Monotonic Learning (WVU)

Monotonic learning is also a method being researched at WVU and by Dr. Ali Mili at the New Jersey Institute of Technology. This method is still in the early stages of development. One issue that must be addressed regarding monotonic learning is how it can be applied to handle larger systems. WVU has some concerns regarding scalability of this method. Since NNs are normally applied to environments that are so complex that other software solutions are impractical, the possible function space that the NN can migrate to during learning may make a monotonic learning monitor unfeasible. The ISR will examine this technique to access its viability and applicability to IVVNN.

6.2.2.3 Safety Monitors

The ISR's experience in the past with developing safety monitors has been in system specific implementations. For example, with the IFC GEN1 program, there are two safety monitors: one for the PTNN and one for the OLNN. The PTNN safety monitor is essentially a reduced table version of the knowledge of the PTNN. This safety monitor resides on a different computer system that has been rigorously tested and has a high level of safety assurance. The monitor checks the outputs from the PTNN to ensure it stays within pre-defined bounds. For the IFC's OLNN, the monitor verifies the range of outputs to ensure they stay within acceptable robustness bounds based upon several different aircraft criteria including physical stress-loads on the F-15.

The ISR will examine the procedures used to develop these safety monitors to establish fundamental concepts that could be used as a guide for run-time monitoring design. Investigation must be done to determine the influence the system's criticality classification has on the level of detail and sophistication of a safety monitor.

6.2.2.4 Lyapunov Stability (NASA DFRC/NASA ARC/WVU)

Lyapunov stability is the current research focus for both NASA DFRC and NASA ARC in regards to run-time monitoring within the IFC program. As a member of this development team, the ISR can conduct additional work to expand the knowledge gained from using Lyapunov stability analysis. Results of this research will discuss function selection, the use of Lyapunov during runtime, and the generalization of this technique beyond the use for an adaptive flight control scheme.

6.2.3 Testing Methods

Currently testing is the first option system developers consider for the assessment of an NN. One technique developed by Taylor and Cukic to improve current testing methods is the automated trajectory generator. The ISR will further refine this test data generator and implement a MATLAB tool that IV&V practitioners can easily use for testing an NN application. The ISR will apply this testing technique towards a study of the IFC program to improve the technique's usefulness.

6.2.4 Visualization Methods

Visualization techniques capitalize on a persons highly developed visual pattern-recognition abilities. During the learning phase and the testing phase visualization techniques may be effectively applied to an NN. Visualization can provide both insight into the decision making process and the learning process of an NN during training. Also, during the testing of a system, a simulation environment like the WVU F-15 Simulation can provide information on the OLNNs ability to adapt in real-time.

6.2.4.1 Visualization of Learning

The ISR will investigate current methods used for the visualization of an NN during the learning phase of its development. Such techniques include the hyperplane animator, trajectory diagrams and visual techniques incorporated in the MATLAB NN Toolbox. Future research in visualization of NN learning must address the challenge of compressing high-dimensional spaces into easily understood and meaningful representations that will give a developer insight into the adaptation of the system during training or operation. The ISR will apply visualization techniques to the NNs of the IFC project to determine their usefulness to the IV&V of the software.

6.2.4.2 WVU F-15 Flight Simulation

The ISR will investigate the success of using a simulation environment for the assessment of NNs with a case study of the WVU F-15 simulation. While this simulation has been specifically created for the IFC program, the effectiveness of using such a system simulation with regard to visualization could be measured. A careful analysis will be done that includes examination of NN parameters across time and examination of the usefulness of visual cues pertaining to system errors in real-time to overall reliability calculations. Results may yield a strategy for increasing the fidelity of a system simulation to build up from a basic NN component level simulation to increasingly higher system level views. The ISR may also be able to identify which levels of fidelity yield better results based upon available resources.

6.3 Tying It All Together

The three stages of the software life cycle where these current methods and techniques appear to have the most impact are development (learning), testing, and operation stages. Once the ISR has studied the above methods in more detail and identified those that are most effective, we will then begin to organize them into a methodology that can be applied to the verification and validation of NNs by the IV&V practitioners.

7.0 REFERENCES

[Aceto 1998]	Aceto, Luca, Augusto Bergueno and Kim G. Larsen. 1998. Model checking via reachability testing for timed automata. In *Proceedings of the 4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* ed. Bernhard Steffen, Gulbenkian Foundation, Lisbon, Portugal, 31 March - 2 April. Lecture Notes in Computer Science 1384:263-280.
[Ackley 1985]	Ackley, D.H., G.E. Hinton, and T.J. Sejnowski. 1985. A learning algorithm for boltzmann machines. *Cognitive Science* 9:147-169.
[Alur 1996]	Alur, R., T.A. Henzinger, and P.H. Ho. 1996. Automatic symbolic verification of embedded systems*. IEEE Transactions on Software Engineering* 22:181-201.
[Andrews 1995a]	Andrews, R., J. Diederich, and A. B. Tickle. 1995. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373-389.
[Andrews 1995b]	Andrews, R. and S.Geva. 1995. RULEX & CEBP networks as the basis for a rule refinement system. In *Hybrid Problems, Hybrid Solutions,* ed. John Hallam. IOS Press. 1-12
[Behrmann 1999]	Behrmann, Gerd, Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. *Efficient timed reachability analysis using clock difference diagrams.* Basic Research in Computer Science Series (BRICS) Report RS-98-47, Department of Computer Science, University of Asrhus, Denmark.
[Bjørner 1996]	Bjørner, Nikolaj, Anca Browne, Eddie Chang, Michael Colón, Arjun Kapur, Zohar Manna, Henny B. Sipma, and Tomás E. Uribe. 1996. STEP: Deductive-algorithmic verification of reactive and real-time systems. *International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Springer-Verlag 1102:415-418.
[Bowman 1998]	Bowman, H., G. Faconti, and M. Massink. 1998. Specification and verification of media constraints using UPPAAL. In *5th Eurographics Workshop on the Design, Specification and Verification of Interactive Systems*, Eurographics Series. Springer-Verlag (August).
[Boyce 1997]	Boyce, W. and DiPrima, R. 1997. *Elementary Differential Equations and Boundary Value Problems.* Sixth Edition. John Wiley & Sons, Inc. ISBN 0-471-08955-9.
[Bradley 1995]	Bradley, S., D. Kendall, W.D. Henderson, and A.P.Robson. 1995. Validation, verification and implementation of timed protocols using AORTA. In *Protocol Specification, Testing and Verification XV (PSTV'95).* Lecture Notes in Computer Science 1066, Springer-Verlag. 208-219.
[Brat 2000]	Brat, Guillaume, K. Havelund, S.J. Park, and W. Visser. 2000. Java Pathfinder: Second generation of a Java model checker. In *Proceedings of the Workshop on Advances in Verification*. Chicago, Illinois.
[Broomhead 1988]	Broomhead, D.S. and D. Lowe. 1988. Multivariable functional interpolation and adaptive networks. *Complex Systems* 2:321-355.
[Buchi 1960]	Buchi, J.R. 1960. Weak second-order arithmetic and finite automata. *Zeitschrift f ur Mathematische Logik und Grundlagen der Mathematik* 6:66-92.

[Cimatti 2002[	Cimatti, Alessandro, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. NuSMV2: An Open Source Tool for Symbolic Model Checking. Technical Report DIT-02-016, Informatica e Telecomunicazioni, Università degli Studi di Trento. Available from the WWW at http://eprints.biblio.unitn.it/archive/00000085/.
[Clarke 2001]	Clarke, E., D. Garlan, B. Krogh, R. Simmons, and J. Wing. 2001. Formal Verification of Autonomous Systems: NASA Intelligent Systems Program. Carnegie Mellon University, Pittsburgh, PA.
[Craven 1992]	Craven, Mark W. and Jude W. Shavlik. 1992. Visualizing learning and computation in artificial neural networks. *International Journal on Artificial Intelligence Tools* 1(3):399-425.
[Craven 1994]	Craven, M. W., and J. W Shavlik. 1994. Using sampling and queries to extract rules from trained neural networks. *Machine Learning: Proceedings of the Eleventh International Conference, San Francisco, CA*. New Brunswick, NJ: Morgan Kaufmann. 37-45.
[Cukic 2001]	Cukic, Bojan. 2001. Levels of fidelity for testing of pre-trained neural networks. Reports provided to the ISR as deliverable for IFCS program in January 2002.
[Cukic 2002]	Cukic, Bojan, Brian J. Taylor, and Harhsinder Singh. 2002. Automated generation of test trajectories for embedded flight control systems. *International Journal of Software Engineering and Knowledge Engineering* 12(2):175-200.
[D'Argenio 1997]	D'Argenio, P.R., J.P. Katoen, T.C. Ruys, and J. Tretmans. 1997. The bounded retransmission protocol must be on time! In *Proceedings of the 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Enschede, The Netherlands, April 1997. Lecture Notes in Computer Science 1217:416-431.
[Daws 1995]	C.Daws and S.Yovine. 1995. Two examples of verification of multirate timed automata with KRONOS. In *Proceedings of the 1995 IEEE Real-Time Systems Symposium, RTSS'95*, Pisa, Italy. IEEE Computer Society Press.
[Duch 2001]	Duch, W., R. Adamczak, and K. Grabczewski. 2001. A new methodology of extraction, optimization, and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks.* 12(2):277-306.
[Dukelow 1994]	Dukelow, J. Verification and validation of neural networks. 1994. Abstract published in *Proceedings of the Neural Network Workshop for the Hanford Community*, Pacific Northwest National Laboratory, Richland, WA. 76-80.
[Fu 1994]	Fu, L. M. Rule generation from neural networks. 1994*IEEE Transactions on Systems, Man, and Cybernetics.* 28(8):1114-1124.
[Gibson 1993]	J. Gibson. 1993. A LOTOS-based approach to neural network specification. Technical Report CSM-112, Department of Computing Science and Mathematics, University of Stirling (May).
[Grossburg 1980]	Grossburg, S. 1980. How does a brain build a cognitive code? *Psychological Review*, 87:1-5.
[Havelund 2000]	Havelund, K., M. Lowry, S. Park, C. Pecheur, J. Penix, W. Visser; and J.L. White. 2000. Formal analysis of the Remote Agent before and after flight. In *Proceedings of 5th NASA Langley Formal Methods Workshop*, Williamsburg, VA. Available from multiple WWW sites, including http://ase.arc.nasa.gov/havelund/Publications/rax.ps.
[Havelund 2001]	Havelund, Klaus and Grigore Rosu. 2001. Java PathExplorer - A runtime verification tool. In *The 6th International Symposium on AI, Robotics and Automation in Space*. Available from the WWW at http://citeseer.nj.nec.com/havelund01java.html

[Haykin 1999]	Haykin, S.. 1999. *Neural Networks: A Comprehensive Foundation.* Second edition. New York: MacMillan Publishing.
[Hebb 1949]	Hebb, D.O. 1949. *The Organization of Behavior: A Neurophysiological Theory.* New York: Wiley
[Henzinger 1992]	Henzinger, T.A., X. Nicollin, J. Sifakis, and S. Yovine. 1992. Symbolic model checking for real-time systems. In *Proceedings of the IEEE Conference on Logics in Computer Science* (LICS).
[Henzinger 1994]	Henzinger, T.A., X. Nicollin, J. Sifakis, and S. Yovine. 1994. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193-244.
[Henzinger 1995]	Henzinger, Thomas A., Pei-Hsin Ho, and Howard Wong-Toi. 1995. A user guide to HyTech. In *Proceedings of the First International Workshop on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS '95), Lecture Notes in Computer Science 1019, Springer-Verlag, 41-71.
[Henzinger 1996]	Henzinger, Thomas A. and Howard Wong-Toi. 1996. Formal methods for industrial applications: Specifying and programming the steam boiler control. Lecture Notes in Computer Science 1165, Springer-Verlag,. 265-282.
[Henzinger 1997a]	Henzinger, Thomas A., Pei-Hsin Ho, and Howard Wong-Toi. 1997. Software tools for technology transfer. In *Proceedings of the Ninth International Conference on Computer-aided Verification* (CAV '97), Lecture Notes in Computer Science 1254, Springer-Verlag, 460-463.
[Henzinger 1997b]	Henzinger, T.A., P.H. Ho, and H. Wong-Toi. 1997. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer* 1:110-122.
[Hietmeyer 1994]	Hietmeyer, C., and N. Lynch. 1994. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proc. ICCC Real-Time Systems Symposium.* IEEE Press. 120-131.
[Hinton 1986]	Hinton, G.E., McClelland, J.L., and Rumelhart, D.E. 1986. Distributed representations. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, eds. D.E. Rumlehart and J.L. McClelland. Cambridge: MIT Press. 77-109.
[Ho 1995]	Ho, Pei-Hsin, and Howard Wong-Toi. 1995. Automated analysis of an audio control protocol. In *Proceedings of the Seventh International Conference on Computer-aided Verification* (CAV 1995), Lecture Notes in Computer Science 939, Springer-Verlag, 381-394.
[Hopfield 1982]	Hopfield, J.J. 1982. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*. 81:3088-3092.
[Jahanian 1995]	Jahanian, Farnam. 1995. Run-time monitoring of real-time systems. In *Advances in Real-Time Systems*, ed. S. Son, Chapter 18. Prentice Hall (out of print). Paper available at http://citeseer.nj.nec.com/chodrow95runtime.html.
[Jensen 1996]	Jensen, Henrik Ejersbo, Kim G. Larsen, and Arne Skou. 1996. Modelling and analysis of a collision avoidance protocol using SPIN and UPPAAL. In *Proceedings of the 2nd SPIN Workshop,* Rutgers University, NJ.
[Jourdan 1993]	Jourdan, M., F. Maraninchi, and A. Olivero. 1993. Verifying quantitative real-time properties of synchronous programs. In *Fifth International Conference on Computer-aided Verfication*, Elounda. Lecture Notes in Computer Science 697, Springer Verlag.
[Keedwell 2001]	Keedwell, E., A Narayanan, and D. Savic. 2001. Creating rules from trained neural networks using genetic alorithms. *International Journal of Computers, Systems, and Signals*. 1:30-43.

[Keller 1996]	Keller, Paul E. Electronic/Artificial Noses Technology Brief, http://lancair.emsl.pnl.gov:2080/proj/neuron//briefs/nose.html.
[Kirkpatrick 1983]	Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi. 1983. Optimization of simulated annealing. *Science*, 220:671-680.
[Knaus 1998]	Knaus, Rodger and Larry Medsker. 1998. *Verification and validation of neural nets.* In *Advanced Research Team '98*, CD-ROM FHWA-RD-99-066 Federal Highway Adminstration, USDOT.
[Kohonen 1982]	Kohonen, T. 1982. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*. 43:49-59.
[Krogh 1995]	Krogh, Anders and Jesper Veldelsby. 1995. Neural network ensembles, cross validation, and active learning. *Advances in Neural Information Processing Systems* 7:231-238.
[Lee 2000]	Lee, Yang-Hang. 2000. An environment for test analysis of real-time software. (Submitted under 2000 Center Software Initiative for the NASA Software IV&V Facility)
[Lisboa 2001]	Lisboa, P. 2001. Industrial use of safety-related artificial neural networks. *Health and Safety Executive Contract Research Report 327.*
[Liu 2002]	Liu, Yan, Tim Menzies, and Bojan Cukic. 2002. Data sniffing – Monitoring of machine learning for online adaptive systems. In *14th IEEE International Conference on Tools with Artificial Intelligence.*
[Mackall 2002]	Mackall, D., S. Nelson, and J. Schumman. 2002. Verification & validation of neural networks for aerospace systems. NASA Technical Report.
[Manna 1991]	Manna, Z. and A. Pnueli. 1991. *The Temporal Logic of Reactive and Concurrent Systems*: *Specification.* NY:Springer-Verlag.
[Manna 1994]	Manna, Z., and A. Pnueli. 1994. Temporal verification diagrams. In *Symposium on Theoretical Aspects of Computer Software,* Lecture Notes in Computer Science 789 Springer-Verlag, 726-765.
[Manna 1995]	Manna, Zohar and the STeP Group. 1995. STeP: The Stanford Temporal Prover (educational release) user's manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University.
[McCulloch 1943]	McCulloch, W.S. and W. Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5:115-133.
[McMillan 1991]	McMillan, C., M. C. Mozer, and P. Smolensky. 1991. The connectionist scientist game: Rule extraction and refinement in a neural network. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ..
[McMillian 2001]	McMillian, K.L. 2001. The SMV system for SMV version 2.5.4. Carnegie Mellon University.
[Menzies 2000]	Menzies, Tim, and Bojan Cukic. 2000. How many tests are enough? *Handbook of Software Engineering and Knowledge Engineering Vol. 2*. World Scientific Publishing Company. ISBN: 981-02-4974-8
[Minksky 1969]	Minsky, M.L., and S.A. Papert. 1969. *Perceptrons*. Cambridge, MA: MIT Press.
[Minsky 1954]	Minsky, M.L. 1954. Theory of neural-analog reinforcement systems and its application to the brain-model problem. Ph.D. thesis, Princeton University, Princeton, NJ.
[Minsky 1967]	Minsky, M.L. 1967. *Computation: Finite and Infinite Machines.* Englewood Cliffs, NJ: Prentice-Hall.
[Misra 1988]	Misra, J., and K. M. Chandy. 1988. *Parallel Program Design: A Foundation.* Addison-Wesley.

[Munro 1991]	Munro, P. 1991. Visualization of 2-D hidden unit space. Technical Report LIS035/IS91003, School of Library and Information Science, University of Pittsburgh, Pittsburgh, PA.
[Neal 1992]	Neal, R.M. 1992. Connectionist learning of belief networks. *Artificial Intelligence* 56:71-113.
[Nelson 2001]	Nelson, Stacy and Charles Pecheur. 2001. *New V&V tools for diagnostic modeling environment(DME)*. Produced for the Space Launch Initiative 2nd Generation RLV TA-5 IVHM Project..
[Nelson 2002]	Nelson, Stacy and Charles Pecheur. 2002. *V&V of advanced systems at NASA*. Produced for the Space Launch Initiative 2nd Generation RLV TA-5 IVHM Project..
[Nicollin 1992]	Nicollin, X., J. Sifakis, and S. Yovine. 1992. Compiling real-time specifications into extended automata. *IEEE TSE Special Issue on Real-Time Systems*, 18(9):794-804.
[NuSMV 2002]	*NuSMV2.1 User Manual.* Available from the WWW at http://nusmv.irst.itc.it/NuSMV/userman/v21/nusmv.pdf.
[Pearl 1988]	Pearl, J. 1988. *Probabilistic reasoning in intelligent systems.* San Mateo, CA: Morgan-Kaufmann.
[Pecheur 1997]	Pecheur, Charles. 1997. Specification and verification of the Co4 distributed knowledge system using LOTOS. In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering*. Incline Village, NV.
[Pecheur 2000]	Pecheur, Charles and Reid Simmons. 2000. From Livingstone to SMV: Formal verification of autonomous spacecraft. In *Proceedings of the FAABS 2000 Conference*, Greenbelt, MD. 103-113.
[Pecheur 2000]	Pecheur, Charles. 2000. *[Projects:] Verification and validation*, Automated Software Engineering Group NASA Ames Research Center Web site, Updated 17 June. Available from the WWW at http://ase.arc.nasa.gov/docs/vandv.html.
[Perhinschi 2002]	Perhinschi, M. G., G. Campa, M. R. Napolitano, M . Lando, L. Massotti, and M.L. Fravolini. 2002. Modeling and simulation of a fault tolerant flight control system. Submitted to *International Journal of Modelling and Simulation* in April 2002.
[Power Technology 1999]	Power Technology. 1999. *Generic NOx control intelligent system*. R&D Brochure. Available from the WWW at http://www.powertech.co.uk/downloads/R&D/NOxControlSystem.pdf.
[Pratt 1991]	Pratt, L.Y. and Mostwo, J. 1991. Direct transfer of learned information among neural networks. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA. 584-589.
[Pratt 1993]	Pratt, L. Y., and Nicodemus, S. 1993. Case studies in the use of a hyperplane animator for neural network research. In *Proceedings of the IEEE International Conference on Neural Networks, IEEE World Congress on Computational Intelligence*, 1:78-83.
[Rodvold 1999]	Rodvold, David M. 1999. A software development process model for artificial neural networks in critical applications. In *Proceedings of the 1999 International Joint Conference on Neural Networks (ICNN'99).*

[Rosenblatt 1960]	Rosenblatt, F. 1960. On the convergence of reinforcement procedures in simple pereceptrons. Cornell Aeronautical Laboratory Report, VG-1196-G-4, Buffalo, NY.
[Rumelhart 1986]	Rumelhart, D.E. and J.L. McClelland, eds., 1986, *Parallel distributed processing: Explorations in the microstructure of cognition*, Vol. 1 & 2, Cambridge, MA: MIT Press.
[Ruys 2002]	Ruys, Theo C. 2002. SPIN beginners' tutorial. Presented at SPIN 2002 Workshop, Grenoble, France April 11. Available on the WWW at http://spinroot.com/spin/Doc/SpinTutorial.pdf.
[Sharkey 1995a]	Sharkey, Amanda and Noel Sharkey. 1995. How to improve the reliability of Artificial Neural Networks. Department of Computer Science, University of Sheffield, UK. Research Report CS-95-11. Available at http://citeseer.nj.nec.com/sharkey95how.html
[Sharkey 1995b]	Sharkey, A.J.C., N.E. Sharkey, and O.G. Chandroth. 1995. Neural nets and diversity. In *Proceedings of the 14th International Conference on Computer Safety, Reliability and Security.* Springer-Verlag. 375-389.
[Simmons 1997]	Simmons, Reid and Gregory Whelan. 1997. Visualization tools for validating software of autonomous spacecraft. In *Proceedings of the 4th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (I-SAIRAS)*, Tokyo, Japan.
[Skapura 1996]	Skapura, David M. and Peter S Gordon. 1996. *Building Neural Networks*. Addison-Wesley.
[Sokolsky 1996]	Sokolsky, Oleg, Insup Lee, and Hanene Ben-Abdallah. 1996. Paragon toolset: A tutorial.
[SPIN 2002]	*On-The-Fly, LTL Model Checking with SPIN.* 2002. Page dated October 1, 2002 available from the WWW at http://spinroot.com/spin/..
[Stolarik 2001]	Stolarik, B. 2001. A neural network-based sensor validation scheme within aircraft control laws. Master's thesis, West Virginia University. Available at http://etd.wvu.edu/ETDS/E2147/Stolarik_Brian_Thesis.pdf
[Taylor 1999]	Taylor, Brian J. 1999. Regressive Model Approach to the Generation of Test Trajectories. Master's thesis, West Virginia University. Available at http://etd.wvu.edu/templates/showETD.cfm?recnum=1077
[Thrun 1995]	Thrun, S. 1995. Extracting rules from artificial neural networks with Distributed Representations. In *Advances in Neural Information Processing Systems (NIPS) 7*, eds G. Tesauro, D. Touretzky, and T. Leen. Cambridge, MA: MIT Press.
[Towell 1990]	Towell, G.G., Shavlik, J.W., and Noordewier, M.O. 1990. Refinement of approximately correct domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. Boston, MA: MIT Press. 861-866
[Towell 1991]	Towell, G.G., M.W. Craven, and J.W. Shavlik. 1991. Constructive induction in knowledge-based neural networks. In *Machine Learning: Proceedings of the Seventh International Workshop.* Evanston, IL. Morgan Kaufmann. 213-217.
[Towell 1993]	Towell, Geoffrey G. and Jude W. Shavlik. 1993. Extracting refined rules from knowledge-based neural networks. *Machine Learning* 13:71-101
[von der Malsburg 1973]	von der Malsburg, C. 1973. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik* 14:85-100
[Wejchert 1990]	Wejchert, J., and Tesauro, G. 1990. Neural network visualization. In *Advances in Neural Information Processing Systems,* Vol. 2, 465-472. San Mateo, CA.: Morgan Kaufmann.

[Widrow 1960]	Widrow, B. and M.E. Hoff, Jr. 1960. Adaptive switching circuits. *IRE WESCON Convention Record.* 96-104.
[Willshaw 1976]	Willshaw, D.J. and C. von der Malsburg. 1976. How patterned neural connections can be set up by self-organization. In *Proceedings of the Royal Society of London Series B* 194:431-445.
[Yu 2002]	Yu, X., O. Efe, O. Kaynak. 2002. A general backpropagation algorithm for feedforward neural network learning. *IEEE Transactions on Neural Networks*, 13(1):251-254.

APPENDIX A – ACRONYMS

AAC	Accurate Automation Corporation
ACRC	Advanced Computing Research Centre
ACSR	Algebra Communicating Shared Resources
ACTIVE	Advanced Control Technology Integrated Vehicles
ARC	Ames Research Center
ART	Adaptive Resonance Theory
ASE	Automated Software Engineering
AVDS	Aviator Visual Design Simulator
BAI	Barron Associates, Inc.
BDD	Binary Decision Diagrams
CADP	Cæsar Aldébaran Development Package
CCS	Calculus Communicating Systems
CEBP	Constrained Error Backpropagation
CMU	Carnegie Mellon University
CSP	Communicating Sequential Processes
CTL	Computation Tree Logic
DCS	Dynamic Cell Structure
DDL	Domain Description Language
DLC	Deterministic Language Construct
DME	Diagnostic Modeling Environment
DS	Deterministic Scope
ESIC	Enhanced Software Instruction Counter
EUCALYPTUS	European/Canadian LOTOS Protocol Tool Set
FHA	Federal Highway Administration
FHWA	Federal Highway Department
GCSR	Graphical Communicating Shared Resources
GLA	Graphical LOTOS Animator
GLD	Graphical LOTOS Designer
GNOCIS	Generic NOx Control Intelligent System
GUI	Graphical User Interface
HSTS	Heuristic Scheduling Testbed System
IEEE	Institute of Electrical and Electronics Engineers
IFC	Intelligent Flight Control
ISO	International Organization for Standardization
ISPP	In-Situ Propellant Production
ISR	Institute for Scientific Research

ITC-IRST	Center for Technological and Scientific Research
IV&V	Independent Verification and Validation
IVVNN	independent verification validation neural networks
JPAX	Java PathExplorer
JPF	Java PathFinder
JPF2	Java PathFinder 2
JVM	Java Virtual Machine
KBANN	Knowledge-Based Neural Network
KBTAC	Knowledge Based Technology Applications Centre
LICS	Logics in Computer Science
LOTOS	Language Of Temporal Ordering Specification
LRU	Locally Responsive Unit
LTL	Linear Temporal Logic
MIR	Mode Identification Recovery
MLP	Multilayer Perceptron
MPL	Modeling Programming Language
NDLC	Non-Deterministic Language Construct
NDS	Non-Deterministic Scope
NN	Neural Network
NNP	Neural Network Processor
NNT	Neural Network Tools
NQC	Not Quite C
OLNN	Online Learning Neural Network
PACSR	Probabilistic ACSR
PARAGON	Process-Algebraic Analysis Real-time Applications with Graphics-Oriented Notation
PC	Program Counter
PNL	Pacific Northwest Laboratory
PROMELA	PROcess MEta Language
PS	Planner Scheduler
PTNN	Pre-Trained Neural Network
PVS	Prototype Verification System
RA	Remote Agent
RAX	Remote Agent eXecutive
RBF	Radial Basis Function
REAL	Rule-Extraction-As-Learning
RIACS	Research Institute Advanced Computer Science
SAT	Satisfiability

SFDIA	Sensor Failure, Detection, Identification, and Accommodation
SIC	Software Instruction Counter
SMILE	SyMbolic Interactive LOTOS Execution
SMV	Symbolic Model Verifier
SOM	Self-Organizing Map
SPIN	Simple Promela Interpreter
SPLICE	Specification and Prototyping with LOTOS for an Interactive Customer Environment
STEP	Stanford Temporal Prover
STTT	Software Tools Technology Transfer
SVC	Stanford Validity Checker
TCTL	Timed Computation Tree Logic
TOPO	Toolset for Product Realization with LOTOS
V&V	Verification and Validation
VIA	Validity Interval Analysis
VERSA	Verification, Execution Rewrite System ACSR
WVU	West Virginia University

APPENDIX B – LIST OF DOCUMENTS

The following documents are included in the ISR V&V of Neural Networks repository classified as relevant or background information:

Abel, Thomas; R. Knauf; and A. Gonzalez. *Generation of a Minimal Set of Test Cases that is Functionally Equivalent to an Exhaustive Set for Use in Knowledge Based System Validation*. Proc. of 9th International Florida Artificial Intelligence Research Symposium 1996 (FLAIRS-96), Key West, FL, USA, 1996, pp. 280-284.

Alexander, Chris; V. Cortellessa; D. Del Gobbo; A. Mili; and M. Napolitano. *Modeling the Fault Tolerant Capability of a Flight Control System: An Exercise in SCR Specification*. 4th NASA Langley Formal Methods Workshop, Williamsburg, VA. June 13-15, 2000.

Andrews, Robert; and S. Geva. *On the Effects of Initializing a Neural Network With Prior Knowledge*. Proceedings of the International Conference on Neural Information Processing (ICONIP'99), Perth Western Australia. 1999. pp251-256.

Andrews, Robert; and S. Geva. *Rule Extraction From Local Cluster Neural Nets*. submitted to Neurocomputing. January 2000.

Andrews, Robert; and S. Geva. *Rules and Local Function Networks*. in "Proceedings of the NIPS*96 Rule Extraction From Trained Artificial Neural Networks Workshop", Andrews R. & Diederich J.(eds). 1996. pp1-12.

Andrews, Robert; and S. Geva. *RULEX & CEBP Networks As the Basis for a Rule Refinement System*. in "Hybrid Problems, Hybrid Solutions", John Hallam (Ed). IOS Press. 1995. pp1-12.

Andrews, Robert; J. Diederich; and A. B. Tickle. *A Survey and Critique Of Techniques For Extracting Rules From Trained Artificial Neural Networks*. Knowledge Based Systems 8. 1995. pp373-389.

Ayache, S.; E. Conquet; Ph. Humbert; C. Rodriguez; J. Sifakis; and R. Gerlich. *Formal Methods for the Validation of Fault Tolerance in Autonomous Spacecraft*. in Proceedings of "The Twenty-Sixth Annual International Symposium on Fault-Tolerant Computing (FTCS '96)", Sendai, Japan. 1996.

Bastani, Farokh B.; and B. Cukic. *A Transformational Approach for Measuring Software Reliability*. The 4th IEEE International Workshop on Evaluation Techniques for Dependable Systems, San Antonio, TX. 1995.

Bastani, Farokh B.; and B. Cukic. *Impact of Program Transformation on Software Reliability Assessment*. IEEE High-Assurance Systems Engineering Workshop, Niagara-on-the-Lake, Ontario, Canada. 1996.

Bastani, Farokh B.; B. Cukic; Hilford; and A. Jamoussi. *Toward Dependable Safety Critical Software*. The 2nd IEEE Workshop on Object-Oriented Real-Time Dependable Systems, Laguna Beach, CA. 1996.

Bedford, D.F.; J. Austin; and G. Morgan. *A Draft Standard for the Certification of Neural Networks used in Safety Critical Systems*. Artifical Neural Networks in Engineering 11/1/1996.

Bedford, D.F.; J. Austin; and G. Morgan. *Requirements for a Standard Certifying the use of Artificial Neural Networks in Safety Critical Applications*. 1/1/1996.

Bolt, George. *Investigating Fault Tolerance in Artificial Neural Networks*. YCS 154, University of York, UK. 1991.

Brat, Guillaume; and W. Visser. *Combining Static Analysis and Model Checking for Software Analysis*. Proceedings of ASE2001. San Diego, CA. 2001.

Brat, Guillaume; K. Havelund; S.J. Park; and W. Visser. *Java Pathfinder - Second Generation of a Java Model Checker*. Proceedings of the Workshop on Advances in Verification. Chicago, Illinois. 2000.

Brown, Steven; and C. Pecheur. *Model-Based Verification of Diagnostic Systems*. Proceedings of JANNAF Joint Meeting, Destin, FL. 2002.

Bryant, Randal E.; E. Clarke; D. Garlan; B. Krogh; R. Simmons; and J. Wing. *Verification Tools for Autonomous and Embedded Systems*. 4/8/2000.

Burton, Simon; J. Clark; A. Galloway; and J. McDermid. *Automated V&V for high integrity systems, a targeted formal methods approach*. In: Proceedings of the LFM 2000, Fifth NASA Langley Formal Methods Workshop. Williamsburg, VA. 2000

Calise, Anthony J.; S. Lee; and M. Sharma. *Development of a Reconfigurable Flight Control Law for the X-36 Tailless Fighter Aircraft*. AIAA Journal of Guidance, Control, and Dynamics, 24(5):896—902. 2001.

Clancey, Daniel; W. Larson; C. Pecheur; P. Engrand; and C. Goodrich. *Autonomous Control of an in-Situ Propellant Production Plant*. In: Proceedings of the Technology 2009 National Conference, Miami Beach, FL. 1999.

Clarke, Edmund; D. Garlan; B. Krogh; R. Simmons; and J. Wing. *Formal Verification of Autonomous Systems NASA Intelligent Systems Program*.. 9/25/2001.

Coenen, Frans; R. Bench-Capon; R. Boswell; J. Dibie-Barthelemy; and B. Eaglestone. *Validation and verification of Knowledge Based Systems: Report on EUROVAV99*. The Knowledge Engineering Review, 15(2):187-196. 2000.

Coit, D.W.; B.T. Jackson; and A.E. Smith. *Static Neural Network Process Models: Considerations and Case Studies*. Journal of Production Research, vol. 36, no. 11, 2953-2967. 1998.

Cortellessa, Vittorio; B. Cukic; A. Mili; D. Del Gobbo; M. Napolitano; M. Shereshevsky; and H. Sanhu. *Certifying Adaptive Flight Control Software*. In: Proceedings of the ISACC 2000 Conference, Reston, VA. 2000.

Craven, Mark; and J.W. Shavlik. *Learning Symbolic Rules Using Artificial Neural Networks*. In: Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology, Bethesda, MD, USA. 1993.

Craven, Mark; and J.W. Shavlik. *Rule Extraction: Where Do We Go from Here?*. University of Wisconsin Machine Learning Research Group Working Paper 99-1. 1999.

Craven, Mark; and J.W. Shavlik. *Understanding Time-Series Networks: A Case Study in Rule Extraction*. International Journal of Neural Systems 8(4): 373-384. 1997.

Craven, Mark; and J.W. Shavlik. *Using Neural Networks for Data Mining*. *Future* Generation Computer Systems (Special Issue on Data Mining) 13:211-229. 1997.

Craven, Mark; and J.W. Shavlik. *Using Sampling and Queries to Extract Rules from Trained Neural Networks*. In: Proceedings of the 11th International Conference on Machine Learning, pp. 37-45, New Brunswick, NJ. 1994.

Craven, Mark; and J.W. Shavlik. *Visualizing Learning and Computation in Artificial Neural Networks*. International Journal on Artificial Intelligence Tools 1(3):399-425. 1992.

Cukic, Bojan; H.H. Ammar; and K. Lateef. *Identifying High-Risk Scenarios of Complex Systems Using Input Domain Partitioning*. The 9th International Symposium on Software Reliability Engineering, Paderborn, Germany. 1998.

Cukic, Bojan; and F.B. Bastani. *Attaining High Confidence in Software Reliability Assessment*. High Integrity Software Conference, Albuquerque, NM. 1997.

Cukic, Bojan; and F.B. Bastani. *Developing Highly-Reliable Software: The MAP Approach*. The 19th Annual International Conference MIPRO '96, Opatija, Croatia. 1996.

Cukic, Bojan. *Accelerated Testing for Software Reliability Assessment*. The 21st Annual International Conference MIPRO '98, Opatija, Croatia. 1998.

Cukic, Bojan. *Combining Testing and Correctness Verification in Software Reliability Assessment*. 2nd IEEE High-Assurance Systems Engineering Symposium, Washington, DC. 1997.

Cukic, Bojan. *Software Design Principles for Improved Reliability Assessment*. The 4th ISSAT International Conference on Reliability and Quality in Design, Seattle, WA. 1998.

Cukic, Bojan. *Transformational Approach to Software Reliability Assessment*. Doctoral Dissertation, Department of Computer Science, University of Houston, Houston, TX. 1997.

Diao, Yizin; and K.M. Passino. *Fault Tolerant Stable Adaptive Fuzzy/Neural Control for a Turbine Engine*. IEEE} Trans. on Control Systems Technology, pp. 494-50. 2001.

Dukelow, James S. Jr. *Verification and Validation of Neural Networks*. Proceedings of the Neural Network Workshop for the Hanford Community, (Ed.s) Paul E. Keller, (Pacific Northwest National Laboratory, Richland, WA, USA, 1994), pp. 76-80. 1/1/1993.

Dutertre, Bruno; and V. Stavridou. *Formal Requirements Analysis of an Avionics Control System*. IEEE Trans. on SE, 23(5):267--278, 1997.

Dwyer, Mathew B.; J. Hatcliff; R. Joehanes; S. Laubach; C.S. Pasareanu; and H. Zheng. *Tools-supported Program Abstraction for Finite-state Verification*. Proceedings of the 23rd International Conference on Software Engineering, May 2001 1/1/2001.

Dybowski, Richard; and S.J. Roberts. *Confidence Intervals and Prediction Intervals for Feed-Forward Neural Networks*. In Dybowski R, Gant V. (eds.) *Clinical Applications of Artificial Neural Networks*. Cambridge: Cambridge University Press. 2001. pp 298-326.

Feather, Martin S.; and B. Smith. *Automatic Generation of Test Oracles - From Pilot Studies to Application*. In: Proceedings of ASE-99: The 14th IEEE Conference on Automated Software Engineering, Cocoa Beach, Florida. IEEE CS Press. 1999.

Fravolini, Mario L.; and M. Napolitano. *A Neural Network Based Tool for Aircraft SFDIA Modeling and Simulation*. IASTED International Conference on Modeling and Simulation, Pittsburgh, PA, USA. 2001.

Garcez, A.S. d'Avila; K. Broda; and D.M. Gabbay. *Symbolic knowledge extraction from trained neural networks: A sound approach*. Artificial Intelligence, 125(1-2):153-205. 2001.

Gobbo, Diego Del; B. Cukic; M. Napolitano; and S. Easterbrook. *Fault Detectability Analysis of Requirements of Validation of Fault Tolerant Systems*. In: Proceedings, Fourth IEEE International Symposium on High Assurance Systems Engineering (HASE'99), Washington DC. 1999.

Gross, Anthony R.; K.R. Sridhar; W. Larson; D. Clancy; C. Pecheur; and G.A. Briggs. *Information Technology and Control Needs for In-Situ Resource Utilization*. In: Proceedings of the 50th IAF Congress, Amsterdam. 1999.

Haley, Pam; D. Soloway; and B. Gold. *Real-time Adaptive Control Using Neural Generalized Predictive Control*. In: Proceedings of the American Control Conference, San Diego, CA. 1999.

Havelund, Klaus; M. Lowry; and J. Penix. *Formal Analysis of a Space Craft Controller using SPIN*. In: Proceedings of the 4th SPIN Workshop, Paris, France. 1999.

Havelund, Klaus; M. Lowry; S. Park; C. Pecheur; J. Penix; W. Visser; and J.L. White. *Formal Analysis of the Remote Agent Before and After Flight*. The Fifth NASA Langley Formal Methods Workshop, Virginia. 2000.

Havelund, Klaus and G. Rosu. *Java PathExplorer – A Runtime Verification Tool.* The 6th International Symposium on AI, Robotics and Automation in Space. 2001. Available from the WWW at http://citeseer.nj.nec.com/havelund01java.html.

Havelund, Klaus and G. Rosu. *Monitoring Java Programs with Java PathExplorer.* RIACS Technical Report 01.19. 2001. Available from the WWW at http://citeseer.nj.nec.com/havelund01monitoring.html.

Hayhurst, Kelly J.; and C.M. Holloway. *Challenges in Software Aspects of Aerospace Systems*. In: Proceedings of the 26th Software Engineering Workshop, Greenbelt, MD. 2001.

Hayhurst, Kelly J.; C.A. Dorsey; J.C. Knight; N.G. Leveson; and G.F. McCormick. *Streamlining Software Aspects of Certification: Report on the SSAC Survey*. NASA/TM-1999-209519. pp. 100. 1999.

Hayhurst, Kelly J.; C.M. Holloway; C.A. Dorsey; J.C. Knight; and N.G. Leveson. *Streamlining Software Aspects of Certification: Technical Team Report on the First Industry Workshop*. NASA/TM-1998-207648. pp. 59. 1998.

Heitmyer, Constance; J. Kirby; and B. Labaw. *Tools for Formal Specification, Verification, and Validation of Requirements*. In: Proceedings of the 12th Annual Conference on Computer Assurance. 1997.

Hinchey, Michael G.; J.L. Rash; and C.A. Rouff. *Verification and Validation of Autonomous Systems*. In M. Hinchey, editor, *Proceedings of 26th Annual NASA Goddard software Engineering Workshop*. pages 136-144, IEEE Computer Society, 2002.

Hovakimyan, Naira; F. Nardi; and A.J. Calise. *A Novel Error Observer Based Adaptive Output Feedback Approach for Control of Uncertain Systems*. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 47, NO. Y, MONTH 2002.

IEEE Software Engineering Standards Committee. *IEEE Standard for Software Verification and Validation*. Inst of Elect & Electronic; ISBN: 0738101966; (February 1986).

Jahanian, Farnam. *Run-Time Monitoring of Real-Time Systems.* Advances in Real-Time Systems, Chapter 18. S. Son, Editor, Prentice Hall. 1995. (Out of Print). Paper available at http://citeseer.nj.nec.com/chodrow95runtime.html.

Jiang, Yuan; Z. Zhou; and Z. Chen. *Rule Learning based on Neural Network Ensemble*. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN'02), Honolulu, HI. Vol.2, pp.1416-1420. 2002.

Johnson, Eric N.; and A.J. Calise. *Neural Network Adaptive Control of Systems with Input Saturation*. In: Proceedings of the American Control Conference, Arlington, Virginia. 2001.

Knauf, Rainer and A. J. Gonzalez. *A Turing Test Approach to Intelligent System Validation*. Wolfgang Wittig und Gunter Grieser (eds): LIT-97, Proc. 5. Leipziger Informatik-Tage, Leipzig, 25./26. pp. 71-76. 1997.

Knauf, Rainer; I. Philippow; and A. Gonzalez. *Towards an Assessment of an AI Systems Validity by a Turing Test*. In: Douglas D. Dankel (ed): Proc. of 10th International Florida Artificial Intelligence Research Symposium 1997 (FLAIRS-97), Daytona Beach, FL, USA, May 11-14, 1997. pp. 397-401, Florida Research Society. 1997.

Kortenkamp, David; R. Simmons; T. Milam; and J.L. Fernandez. *A Suite of Tools for Debugging Distributed Autonomous Systems*. Proceedings of the IEEE International Conference on Robotics and Automation, Washington DC, May 2002.

Krogh, Bruce; and J. Vedelsby. *Neural Network Ensembles, Cross Validation, and Active Learning*. In G Tesauro, D S Touretzky, and T K Leen (eds.), 231-238. 1995.

Lawrence, Steve; A. Black; A.C. Tsoi; and C.L. Giles. *On the Distribution of Performance from Multiple Neural-Network Trials*. IEEE Transactions on Neural Networks (IEEE TNN), Vol. 8, No. 6, pp. 1507 - 1517. 1997.

Lee, Dr. Yann-Hang. *RTESV&V Report Program Analyzer to Establish Models and Instrumented Codes*. 8/1/2000.

Lee, Dr. Yann-Hang. *RTESV&V Report V&V Tool Suite for Real Time Control Systems*.. 7/7/2000.

Lee, Yann-Hang. *An Environment for Test Analysis of Real-Time Software*. 1/22/2000.

Leitner, Jesse; A. Calise; and J.V.R. Prasad. *Analysis of Adaptive Neural Networks for Helicopter Flight Controls*. In: Proceedings of the AIAA Guidance, Navigation, and Control Conference, Baltimore, MD. pp. 871-879. 1995.

Lerda, Flavio; and W. Visser. *Addressing Dynamic Issues of Program Model Checking*. In: Proceedings of the 8th international SPIN workshop on Model checking of software, Toronto, Ontario, Canada. p.80-102. 2001.

Liu; T. Menzies; and B. Cukic. *Data Sniffing – Monitoring of Machine Learning for Online Adaptive Systems*. 14th IEEE International Conference on Tools with Artificial Intelligence November 4-6, 2002

Loparo, Kenneth A. *Lyapunov Stablility Analysis of Dynamical Systems*.7/1/2002.

Lowe, David; and K. Zapart. *Point-Wise Confidence Interval Estimation by Neural Networks: A Comparative Study Based on Automotive Engine Calibration*. Neural Computing & Applications Volume 8 Issue 1 (1999) pp 77-85.

Lowry, Michael; and D. Dvorak. *Analytic Verification of Flight Software*. In *IEEE Intelligent Systems* 13(5): 45-49. 1998.

Lowry, Michael; K. Havelund; and J. Penix. *Verification and Validation of AI Systems that Control Deep-Space Spacecraft*. 1997. Available on the WWW at http://ase.arc.nasa.gov/papers/ISMIS97/ISMIS97.Revised.pdf.

Mackall, Dale; S. Nelson; and J. Schumman. *Verification & Validation of Neural Networks for Aerospace Systems*. NASA Ames Research Center. 6/12/2002.

Menzies, Tim; and B. Cukic. *Adequacy of Limited Testing for Knowledge Based Systems*. International Journal on Artificial Intelligence Tools 9(1): 153-172. 2000.

Menzies, Tim; and B. Cukic. *Average Case Coverage for Validation of AI Systems*. *AAAI* Stanford Spring Symposium on Model-based Validation of AI Systems. 2001. Available on the WWW at http://tim.menzies.com/pdf/01validint.pdf.

Menzies, Tim; and B. Cukic. *How Many Tests Are Enough*. Handbook of Software Engineering and Knowledge Engineering, Volume II; Editor S.K. Chang; 981-02-4974-8. 2002. Available on the WWW at http://tim.menzies.com/pdf/00ntests.pdf.

Menzies, Tim; and B. Cukic. *Intelligent Testing can be Very Lazy*. In: Proceedings of the AAAI '99 workshop on Intelligent Software Engineering, Orlando, Florida. 1999. Available on the WWW at http://tim.menzies.com/pdf/99waaai.pdf.

Menzies, Tim; and B. Cukic. *When to Test Less*. IEEE Software, pp. 107-112, volume 17, number 5. 2000. Available on the WWW at http://tim.menzies.com/pdf/00iesoft.pdf.

Menzies, Tim; and P. Compton. *The (Extensive) Implications of Evaluation on the Development of Knowledge-Based System*. Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge Based Systems 1/1/1995.

Menzies, Tim; and S. Waugh. *Lower Bounds on the Size of Test Data Sets*. In: Proceedings of the Australian Joint Conference on Artificial Intelligence, pp. 227-237. 1998.

Menzies, Tim; and Y. Hu. *Just Enough Learning (of Association Rules)*. WVU CSEE tech report. 2002. Available on the WWW at http://tim.menzies.com/pdf/02tar2.pdf.

Menzies, Tim; B. Cukic; H. Singh; and J. Powell. *Testing Nondeterminate Systems*. In: Proceedings of the ISSRE 2000. 2000. Available on the WWW at http://tim.menzies.com/pdf/00issre.pdf.

Menzies, Tim; S. Easterbrook; B. Nuseibeh; and S. Waugh. *Validating-Inconsistent Requirements Models using Graph-based Abduction*. http://www.cs.toronto.edu/%7Esme/papers/2000/abduction.pdf 1/1/2000.

Menzies, Tim; D. Owen; and B. Cukic. *Saturation Effects in Testing of Formal Models*. In: Proceedings of the ISSRE 2002. 2002. Available on the WWW at http://tim.menzies.com/pdf/02sat.pdf.

Menzies, Tim. *Verification and Validation and Artificial Intelligence*. Foundations 2002, October 22-23, 2002.

Mitchell, Tom M.; and S.B. Thrun. *Learning Analytically and Inductively*. Mind Matters: A Tribute to {A}llen {N}ewell. Lawrence Erlbaum Associates, Inc. 85--110. 1996.

Morgan, G.; J. Austin; and G. Bolt. *Safety Critical Neural Networks*. IEEE Conference on Neural Networks, 1995.

Nardi, Flavio; N. Hovakimyan; and A.J. Calise. *Decentralized Control of Large-Scale Systems using Single Hidden Layer Neural Networks*. In: Proceedings of the 2001 American Control Conference. 2001.

NASA ARC; S. Nelson; and C. Pecheur. *New V&V Tools for Diagnostic Modeling Environment(DME)*. Task 10 TA-5.3.3(WBS 1.4.4.5.3) of the Space Launch Initiative 2nd Generation RLV TA-5 IVHM Project. 1/25 2001.

NASA ARC; S. Nelson; and C. Pecheur. *V&V of Advanced Systems at NASA*. Task 10 TA-5.3.3(WBS 1.4.4.5.3 of the Space Launch Initiative 2nd Generation RLV TA-5 IVHM Project. 1/25/2002.

NASA ARC; S. Nelson; and C. Pecheur. *Survey of NASA V&V Processes/Methods for Northrop Grumman Corp*. NASA ARC TASK NO: 10 TA-5.3.3 (WBS 1.4.4.5.3) 10/24/2001.

Naydich, Dimitri; and J. Nowakowski. *Flight Guidance System Validation using SPIN*. NASA Contractor Report NASA/CR-1998-208434. 1998

Omlin, Christian W.; and C.L. Giles. *Rule Revision with Recurrent Neural Networks*. TKDE 8(1): 183-188. 1996.

Ordonez, Raul; and K.M. Passino. *Stable Multi-Input Multi-Output Adaptive Fuzzy/Neural Control*. IEEE TRANSACTIONS ON FUZZY SYSTEMS, VOL. 7, NO. 3, JUNE 1999

Pasareanu, Corina S.; M.B. Dwyer; and W. Visser. *Finding Feasible Counter-Examples When Model Checking Abstracted Java Programs*. Lecture Notes in Computer Science. Vol. 2031, pp. 284 --. 2001.

Pecheur, Charles; and A. Cimatti. *Formal Verification of Diagnosability via Symbolic Model Checking*. In: Proceedings of the MoChArt 2002 Workshop. Lyon, France. 2002.

Pecheur, Charles; and R. Simmons. *From Livingstone to SMV Formal Verification for Autonomous Spacecrafts*. In: Proceedings of the FAABS 2000 Conference. Greenbelt, Maryland. 2000.

Pecheur, Charles. *Advanced Modeling and Verification Techniques Applied to a Cluster File System*. In: proceedings of the ASE'99 Conference. Cocoa Beach, Florida. 1999.

Pecheur, Charles. *Specification and Verification of the Co4 Distributed Knowledge System Using LOTOS*. In: Proceedings of the 12th IEEE International Conference on Automated Software Engineering. Incline Village, Nevada. 1997.

Pecheur, Charles. *Verification and Validation of Autonomy Software at NASA*. NASA/TM 2000-209602. 2000.

Pecheur, Charles. *[Projects:] Verification and Validation*. Automated Software Engineering Group NASA Ames Research Center Web site. Updated 6/17/2000. Web page available from WWW at http://ase.arc.nasa.gov/docs/vandv.html.

Pecheur, Charles; W. Visser; and R. Simmons. *RIACS Workshop on the Verification and Validation of Autonomous and Adaptive Systems*. Conference report, AI Magazine, Fall 2001. Also available as NASA/TM-2001-210927. 2001.

Pell; Gat; Keesing; Muscettola; and B. Smith. *Robust Periodic Planning and Execution for Autonomous Spacecraft*. Proceedings of IJCAI-97. 1997. Available from the WWW at http://ic.arc.nasa.gov/ic/projects/Executive/papers/ijcai97.ps.

Penix, John; C. Pecheur; and K. Havelund. *Using Model Checking to Validate AI Planner Domain Models*. In: Proceedings of the 23rd Annual Software Engineering Workshop. NASA Goddard. 1998.

Penix, John; W. Visser; E. Engstrom; A. Larson; and N. Weininger. *Verification of Time Partitioning in the DEOS Scheduler Kernel*. International Conference on Software Engineering, pp. 488-497. 2000.

Preece, Alun D. *Validation of Knowledge-Based Systems: Current Trends and Issues*. *Journal of Communication and Cognition - Artificial Intelligence*, 11(4), 381-413, 1994.

Rosenbert, Linds; L. Hyatt; T. Hammer; L. Huffman; and W. Wilson. *Testing Metrics for Requirement Quality*. Quality Week Europe '98 Conference, November, 1998, Belguim 11/1/1998.

Sethi, Ishwar K.; and J.H. Yoo. *Multivalued Logic Mapping of Neurons in Feedforward Networks*. AAAI-96.

Simmons, Reid; and C. Pecheur. *Automating Model Checking for Autonomous Systems*. In: Proceedings of AAAI Spring Symposium on Real-Time Autonomous Systems. Stanford. 2000.

Simmons, Reid; and G. Whelan. *Visualization Tools for Validating Software of Autonomous Spacecraft*. In: Proceedings of the Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS). Tokyo, Japan. 1997.

Simmons, Reid; C. Pecheur; and G. Srinivasan. *Towards Automatic Verification of Autonomous Systems*. In: Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE. 2000.

Smith, Ben; W. Miller; J. Dunphy; Y. Tung; P. Nayak; E. Gamble; and M. Clark. *Validation and Verification of the Remote Agent for Spacecraft Autonomy*. IEEE Aerospace 1999.

Thrun, Sebastian B. *Extracting Provably Correct Rules from Artificial Neural Networks*. Technical Report IAI-TR-93-5, Institute fur Informatics III, Universität Bonn. 1993.

Tickle, Alan B.; R. Andrews; M. Golea; and J. Diederich. *The truth is in there: directions and challenges in extracting rules from trained artificial neural networks*. 1/1/1998.

Towell, Geoffrey G.; and Jude W. Shavlik. *Extracting Refined Rules from Knowledge-Based Neural Networks*. Machine Learning (in press). 1991.

Visser, Willem; and H. Baringer. *Memory Efficient State Storage in SPIN*. In: Proceedings of SPIN96 Workshop. Rutgers, USA. 1996.

Visser, Willem; and H. Baringer. *Practical CTL* Model Checking -- Should SPIN be Extended?*. International Journal on Software Tools for Technology Transfer (STTT). Volume 2 Number 4. 2000.

Visser, Willem; H. Baringer; D. Fellows; G. Gough; and A. Williams. *Efficient CTL* Model Checking for Analysis of Rainbow Designs*. In: Proceedings of CHARME97 Workshop. Montreal, Canada. 1997.

Visser, Willem; G. Brat; K. Havelund; and S. Park. *Model Checking Programs*. In: Proceedings of the 15th International Conference on Automated Software Engineering (ASE). Grenoble, France. 2000.

Visser, Willem; K. Havelund; and J. Penix. *Adding Active Objects to SPIN*. In: Proceedings of SPIN99a Workshop. Trento, Italy. 1999.

Wen, Wu; and J. Callahan. *Neuralware Engineering: Develop Verifiable ANN-based Systems*. In Symposium on Intelligent Systems and Robotics, Washington D. C. 1996.

Wen, Wu; and J. Callahan. *Verification and Validation of KBS With Neural Network Components*. AAAI 1996.

Wen, Wu; J. Callahan; and M. Napolitano. *Towards Developing Verifiable Neural Network Controller*. In: Proceedings of ICTAI'96, pp 75—82. Toulous, France. 1996.

Wen, Wu; J. Callahan; and M. Napolitano. *Verifying Stability of Dynamic Soft-Computing Systems*. In: Proceedings of IJCAI Workshop on Validation, Verification and Refinement of AI Systems and Subsystems. Nagoya, Japan. pp 11-18. 1997.

Williams, Christopher K.I.; and F. Vivarelli. *Upper and Lower Bounds on the Learning Curve for Gaussian Processes*. Machine Learning 40(1): 77-102. 2000.

Yang, Bwolan; R. Simmons; D.R. O'Hallaron; and R.E. Bryant. *Optimizing Symbolic Model Checking for Constraint-Rich Models*. CAV. 328-340. 1999.

Zhou, Zhi-Hua; Jiang; and S. Chen. *Extracting Symbolic Rules from Trained Neural Network Ensembles*. AI Communications, in press. 2002.